Reti Neurali in Generale

Le Reti Neurali Artifi ciali sono studiate sotto molti punti di vista. In particolare, contributi alla ricerca in questo campo provengono da:

- Biologia (Neurofi siologia)
- Informatica (Intelligenza Artifi ciale)
- Matematica (Ottimizzazione, Proprietà di approssimazione)
- Statistica (Regressione, Classifi cazione)
- Ingegneria (Pattern Classifi cation, Teoria del Controllo)
- Economia (Studio di serie temporali)
- Fisica (Sistemi Dinamici)
- Psicologia (Apprendimento e Scienze Cognitive)

Reti Neurali in Generale

Bisogna distinguere due motivazioni diverse nello studiare Reti Neurali Artifi ciali:

1. riprodurre (e quindi comprendere) il cervello umano
   - modellare tutto o almeno parti del cervello umano in modo affi dabile
   - riprodurre fedelmente fenomeni neurofi siologici
   - verifi care sperimentalmente se il modello proposto riproduce i dati biologici
2. estrarre i principi fondamentali di calcolo utilizzati dal cervello
   - non importa riprodurre il cervello umano, ma solo evincere quali sono i principi fondamentali di calcolo che esso utilizza
   - semplifi cazioni ed astrazioni del carvello umano sono permesse, anzi, sono lo strumento principale di lavoro
   - produrre un sistema artifi ciale che sia eventualmente diverso dal cervello umano ma che riproduca alcune delle sue funzioni, magari in modo pi`u veloce ed effi ciente (metafora del volo: aereo “contro” uccello)

Reti Neurali in Generale

A noi interessa la seconda motivazione.

I modelli di Reti Neurali Artifi ciali proposti in questo ambito sono molteplici e con scopi diversi.

As esempio, esistono modelli per

- l'apprendimento supervisionato (Classifi cazione, Regressione, Serie Temporali, ...);
- l'apprendimento non supervisionato (Clustering, Data Mining, Self-Organization maps,...);
- realizzare Memorie Associative;
- ...

Tali modelli, in generale, differiscono per

- topologia della rete
- funzione calcolata dal singolo neurone
- algoritmo di apprendimento
- modalità di apprendimento (utilizzo dei dati di apprendimento)

Reti Neurali in Generale

Quando è opportuno utilizzare una Rete Neurale Artifi ciale ?

- l’input è ad alta dimensionalità (discreto e/o a valori reali)
- l’output è a valori discreti (classifi cazione) o a valori reali (regressione)
- l’output è costituito da un vettore di valori
- i dati possono contenere rumore
- la forma della funzione target è totalmente sconosciuta
- la soluzione fi nale non deve essere compresa da un esperto umano (“black-box problem”)

Esempi di campi applicativi

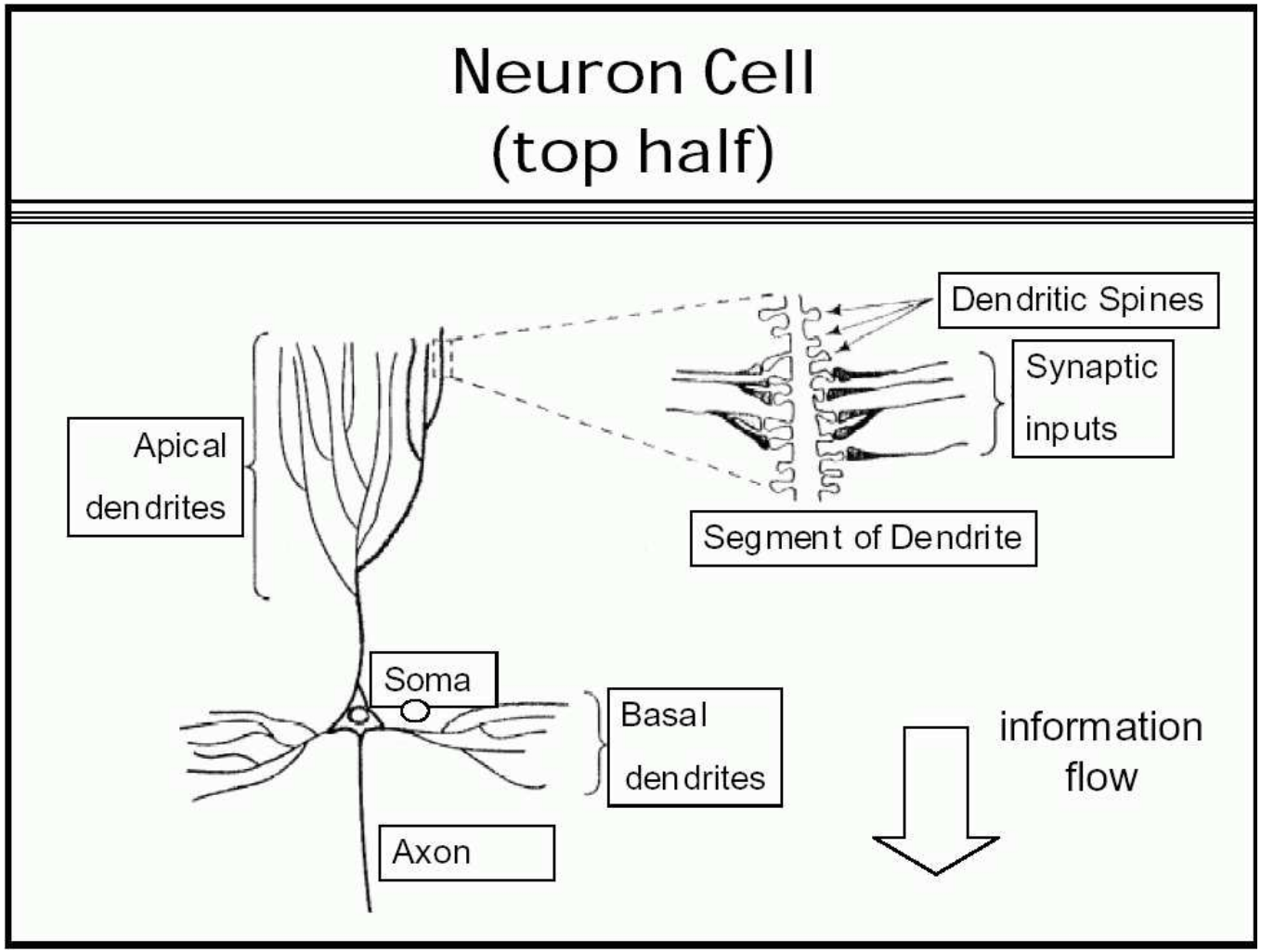
- riconoscimento del parlato
- classifi cazione di immagini
- predizione di serie temporali in fi nanza
- controllo di processi industriali

Reti Neurali Artifi ciali

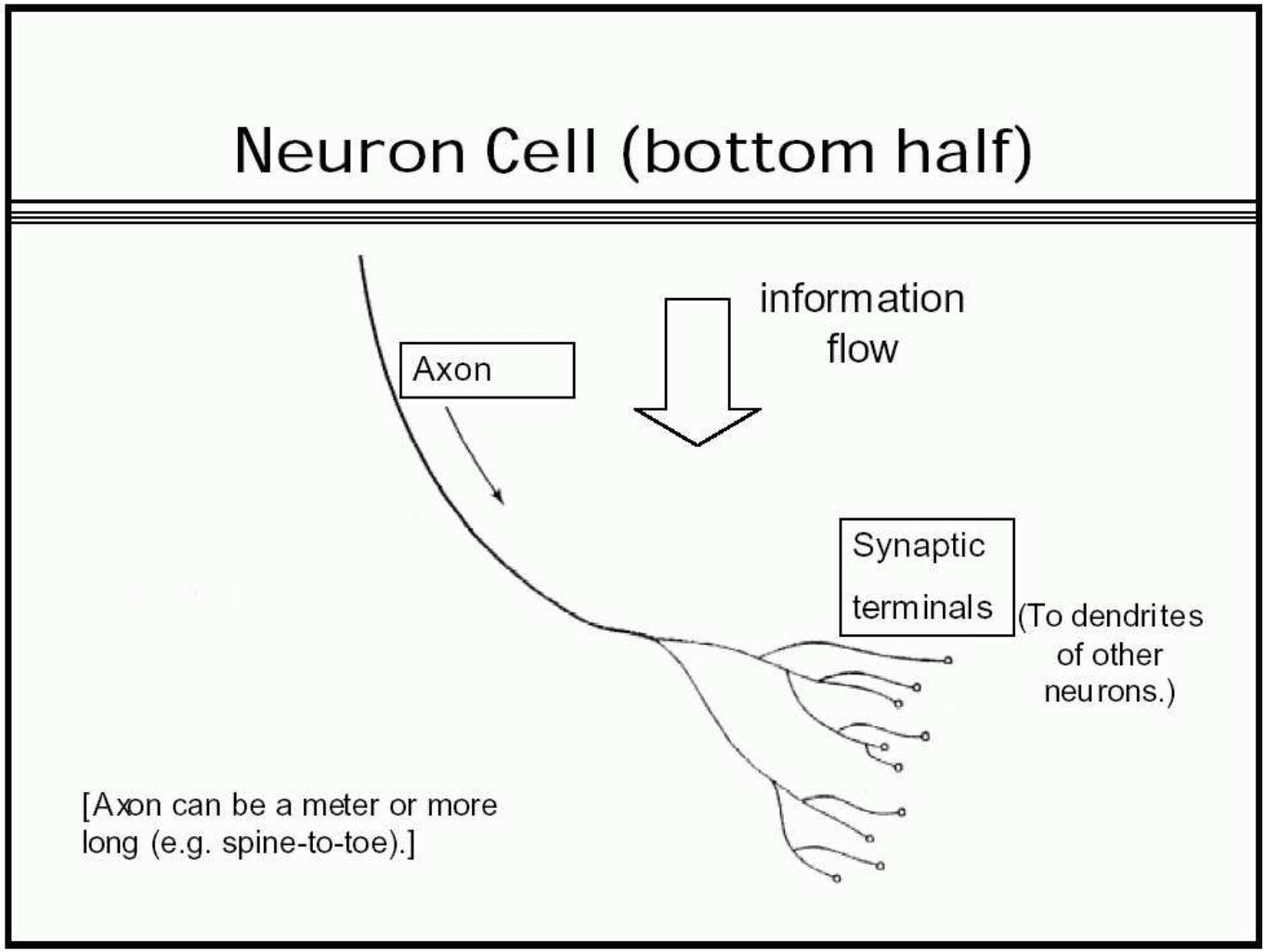
Le Reti Neurali Artificiali si ispirano al cervello umano:

- Il cervello umano è costituito da circa 10^{10} neuroni fortemente interconnessi fra loro;
- Ogni neurone possiede un numero di connessioni che va da circa 10^4 a circa 10^5;
- Il tempo di risposta di un neurone è circa 0.001 secondi;
- Considerando che per riconoscere il contenuto di una scena un umano impiega circa 0.1 secondi, ne consegue che il cervello umano sfrutta pesantemente il calcolo parallelo: infatti, in questo caso, non può effettuare più di 100 calcoli seriali [0.1/0.001=100].

Neurone Biologico

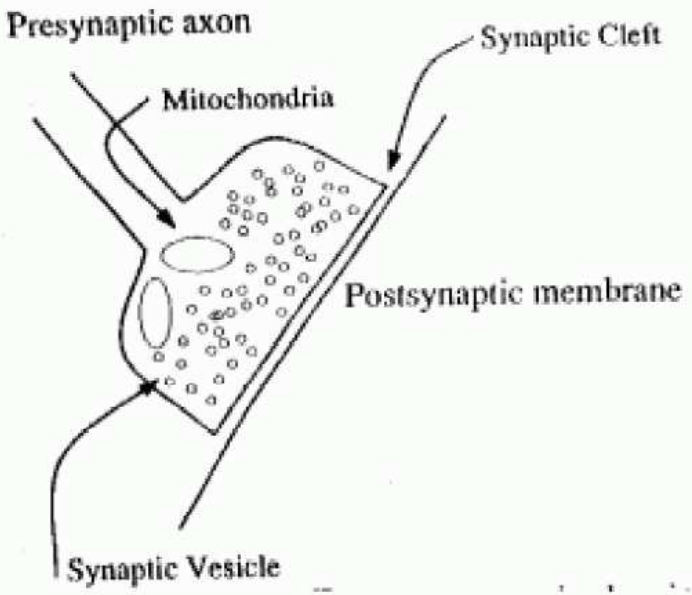


Neurone Biologico



Neurone Biologico

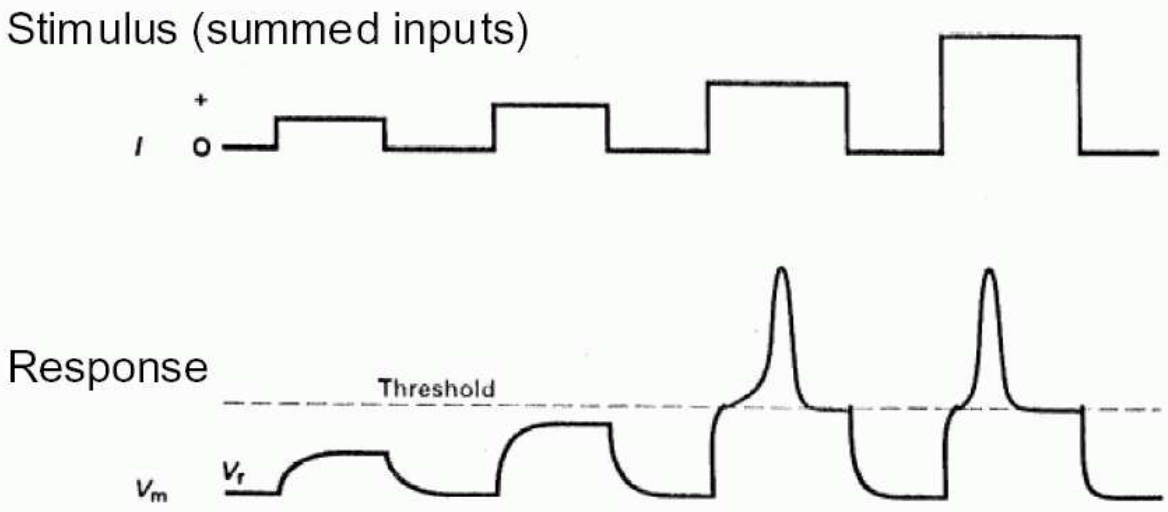
Chemical Synapse



reference: James A. Anderson, An Introduction to Neural Networks, MIT Press, 1955.

Neurone Biologico

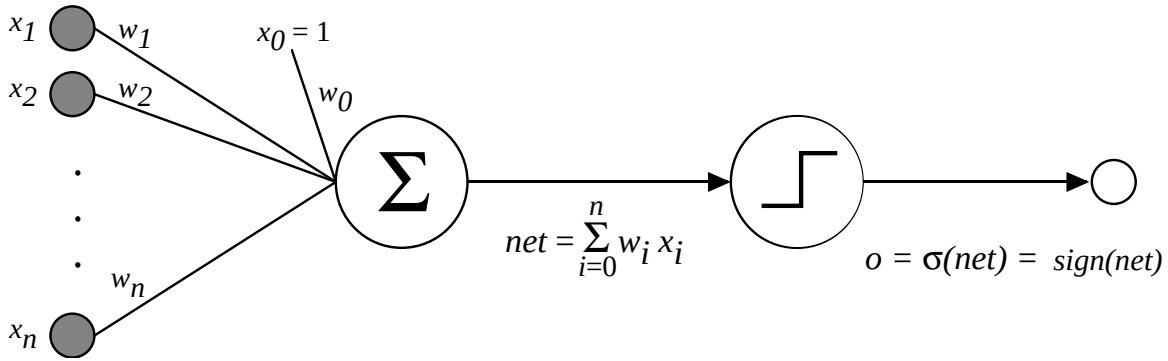
Triggering phenomenon



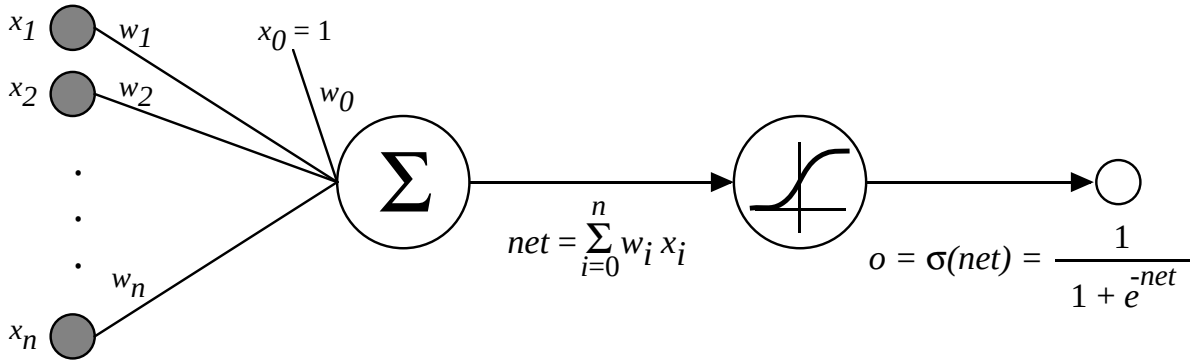
reference: Irwin B. Levitan and Leonard K. Kaczmarek, The Neuron, Oxford University Press, 1991.

Neurone Artifi ciale

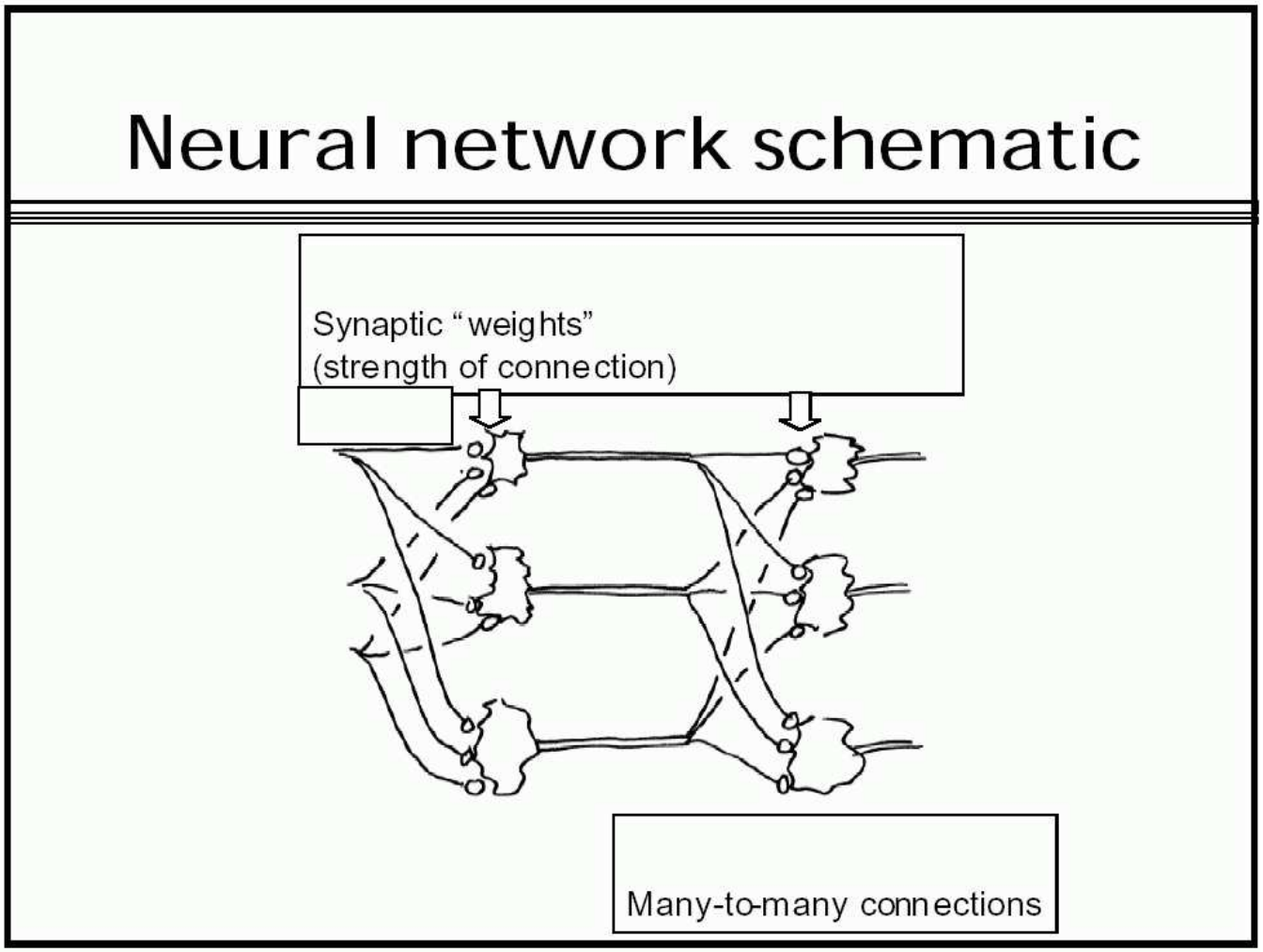
Alternativa 1: hard-threshold $\rightarrow$ iperpiano!!



Alternativa 2: neurone sigmoidale $\rightarrow$ funzione derivabile



Rete Neurale Biologica (schema)

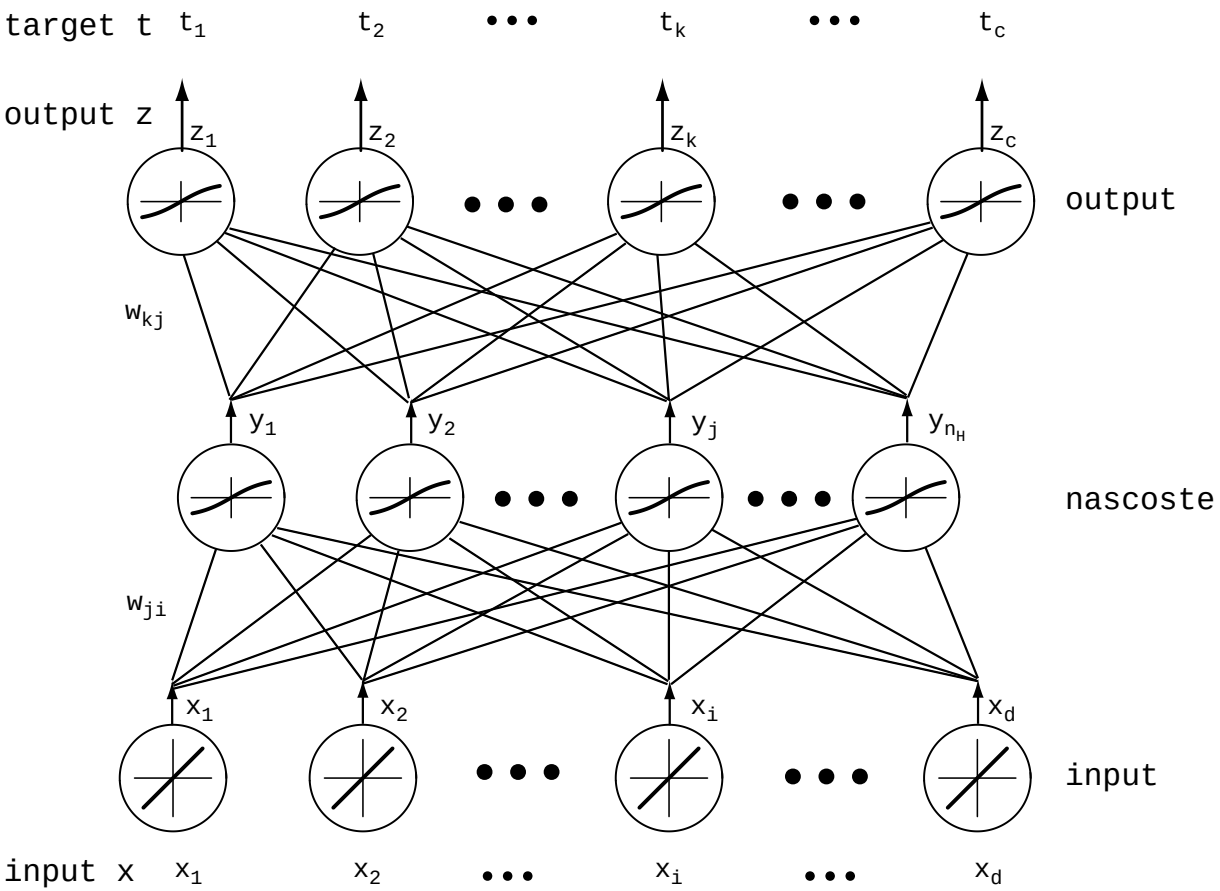


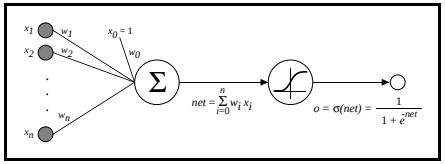
Reti Neurali Artifi ciali

Una Rete Neurale Artificiale è un sistema costituito da unità interconnesse che calcolano funzioni (numeriche) non-lineari:

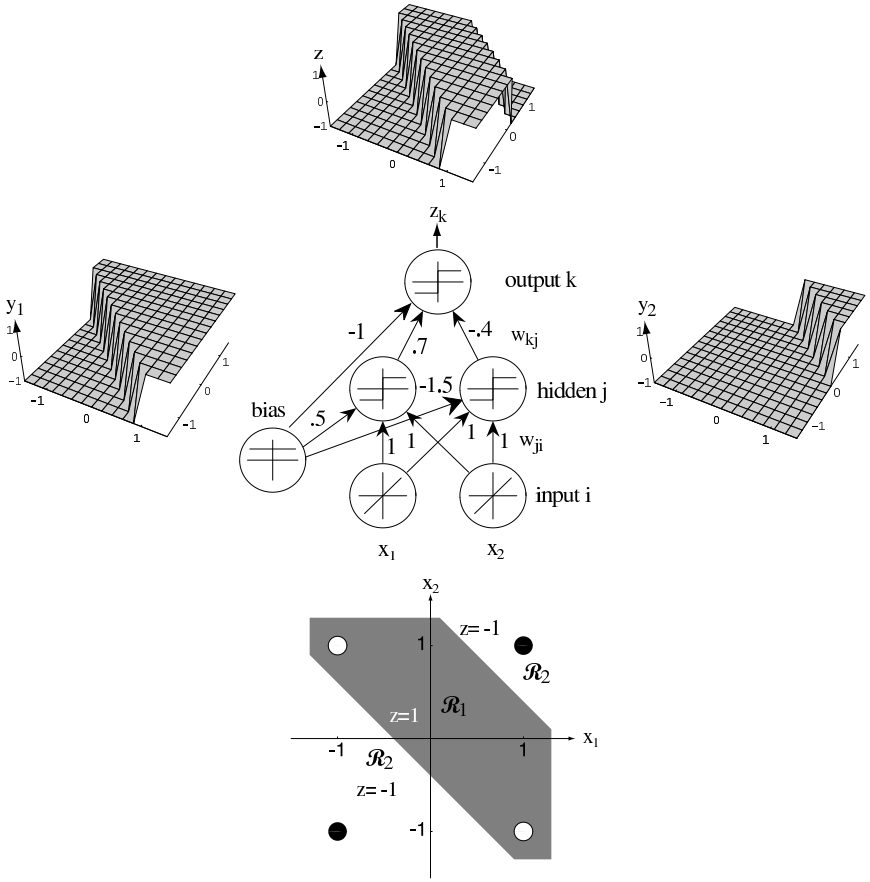
- le unità di *input* rappresentano le variabili di ingresso;
- le unità di *output* rappresentano le variabili di uscita;
- le unità di *nascoste* (se ve ne sono) rappresentano variabili interne che codificano (dopo l'apprendimento) le correlazioni tra le variabili di input relativamente al valore di output che si vuole generare.
- sulle connessioni fra unità sono definiti *pesi* adattabili (dall'algoritmo di apprendimento).

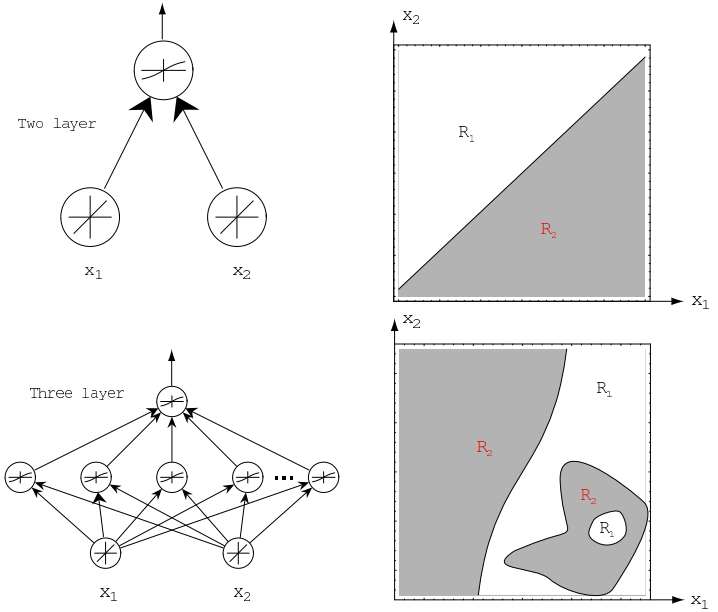
Reti Neurali Feed-forward (un solo livello nascosto)





Reti Neurali Feed-forward e Superfi ci di Decisione





Singolo Neurone con Hard-Threshold

Singolo Neurone con Hard-Threshold: iperpiani!

$$\mathcal{H} = \{f_{(\vec{w},b)}(\vec{y}) | f_{(\vec{w},b)}(\vec{y}) = sign(\vec{w} \cdot \vec{y} + b), \vec{w}, \vec{y} \in \mathbb{R}^n, b \in \mathbb{R}\}$$

che possiamo riscrivere come

$$\mathcal{H} = \{f_{(\vec{w}')}(\vec{y}') | f_{(\vec{w}')}(\vec{y}') = sign(\vec{w}' \cdot \vec{y}'), \vec{w}', \vec{y}' \in \mathbb{R}^{n+1}\}$$

se effettuiamo le segenti trasformazioni

$$\vec{w}' = [b, \vec{w}]^t \qquad \vec{y}' = [1, \vec{y}]^t$$

Faremo riferimento a tale neurone (e all'algoritmo di apprendimento associato) come Perceptron

Problemi: Quale è il potere computazionale di un Perceptron ?

Come definire un algoritmo di apprendimento ?

Funzioni Booleane

Consideriamo input binari e funzioni booleane.

Ogni funzione booleana può essere rappresentata tramite gli operatori or,and, not (in effetti, questo non è necessario se usiamo il nand o il nor)

Può un Perceptron implementare l'operatore or ?

Funzioni Booleane

Consideriamo input binari e funzioni booleane.

Ogni funzione booleana può essere rappresentata tramite gli operatori or,and, not (in effetti, questo non è necessario se usiamo il nand o il nor)

Può un Perceptron implementare l'operatore or ? Si !

Es. $\vec{y}' \in \{0,1\}^{n+1}, w'_0 = -0.5, w'_i = 1,\ i = 1,\ldots,n$

Può un Perceptron implementare l'operatore and ?

Funzioni Booleane

Consideriamo input binari e funzioni booleane.

Ogni funzione booleana può essere rappresentata tramite gli operatori or,and, not (in effetti, questo non è necessario se usiamo il nand o il nor)

Può un Perceptron implementare l'operatore or ? Si !

Es. $\vec{y}' \in \{0,1\}^{n+1}, w'_0 = -0.5, w'_i = 1,\ i = 1,\ldots,n$

Può un Perceptron implementare l'operatore and ? Si !

Es. $\vec{y}' \in \{0,1\}^{n+1}, w'_0 = -n + 0.5, w'_i = 1,\ i = 1,\ldots,n$

Funzioni Booleane

Consideriamo input binari e funzioni booleane.

Ogni funzione booleana può essere rappresentata tramite gli operatori or,and, not (in effetti, questo non è necessario se usiamo il nand o il nor)

Può un Perceptron implementare l'operatore or ? Si !

Es. $\vec{y}' \in \{0,1\}^{n+1}, w'_0 = -0.5, w'_i = 1,\ i = 1,\ldots,n$

Può un Perceptron implementare l'operatore and ? Si !

Es. $\vec{y}' \in \{0,1\}^{n+1}, w'_0 = -n + 0.5, w'_i = 1,\ i = 1,\ldots,n$

L'operatore not si realizza banalmente con un Perceptron con una singola connessione (fare per esercizio).

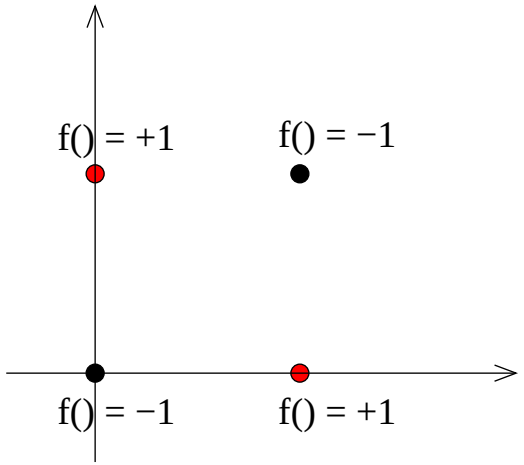
Esiste una funzione booleana semplice che non può essere realizzata da un Perceptron ?

(l'abbiamo già vista quando abbiamo calcolato la VC-dimension di $\mathcal{H}$!!)

Funzione non-linearmente separabili

XOR: funzione non linearmente separabile !

input	XOR
0 0	$false$
0 1	$true$
1 0	$true$
1 1	$false$



una funzione è linearmente separabile se esiste un iperpiano che separa gli ingressi positivi (+1) da quelli negativi (-1)

Apprendimento di funzioni linearmente separabili

Assumiamo di avere esempi di funzioni linearmente separabili.

Algoritmo di Apprendimento per il Perceptron

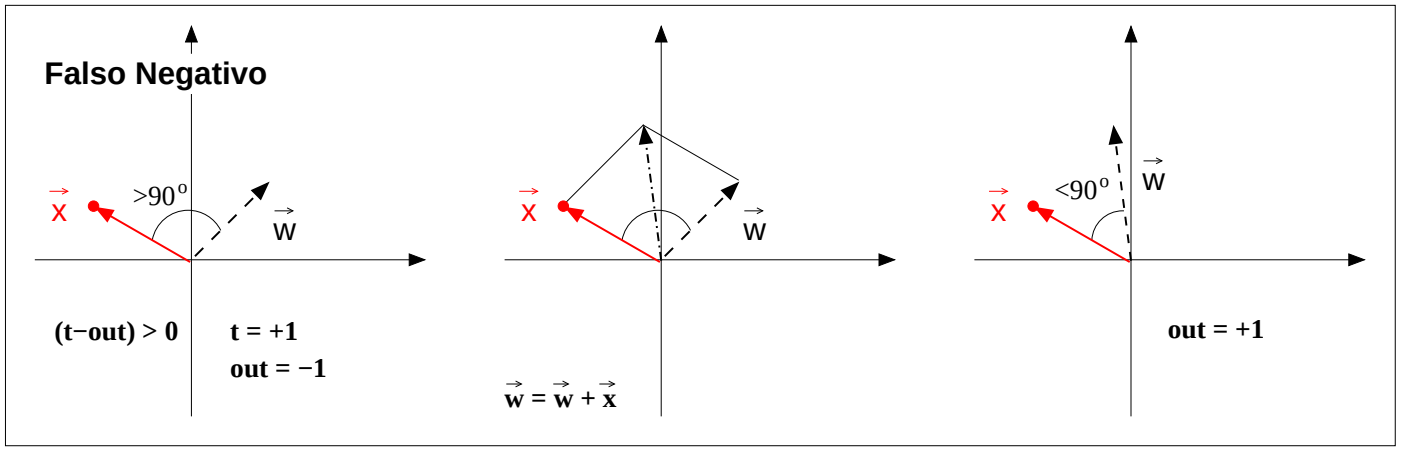
ingresso: insieme di apprendimento $Tr = \{(\vec{x}, t)\}$, dove $t \in \{-1, +1\}$

1. inizializza il vettore dei pesi $\vec{w}$ al vettore nullo (tutte le componenti a 0);
2. ripeti
   (a) seleziona (a caso) uno degli esempi di apprendimento $(\vec{x}, t)$
   (b) se $out = sign(\vec{w} \cdot \vec{x}) \neq t$ allora

$$\vec{w} \leftarrow \vec{w} + (t - out)\vec{x}$$

Apprendimento per Perceptron

Interpretazione geometrica



Apprendimento per Perceptron

Non necessariamente un singolo passo di apprendimento 2(b) riuscirà a modifi care il segno dell'output: potrebbero servire molti passi.

Per rendere pi`u stabile l'apprendimento si aggiunge un coeffi ciente di apprendimento η

$$\vec{w} \leftarrow \vec{w} + \eta(t - out)\vec{x}$$

con $\eta > 0$ e preferibilmente $\eta < 1$

Questo evita che il vettore dei pesi subisca variazioni troppo "violente" ogni volta che il passo 2(b) viene eseguito, e quindi cerca di evitare che esempi precedentemente ben classifi cati diventino classifi cati erroneamente a causa della forte variazione del vettore dei pesi.

Se l'insieme di apprendimento è LINEARMENTE SEPARABILE, si dimostra che l'algoritmo di apprendimento per il Perceptron termina con una soluzione in un numero fi nito di passi, altrimenti $\vec{w}$ CICLA in un insieme di vettori peso non necessariamente ottimali (cioè che commettono il minimo numero possibile di errori)

Apprendimento per Perceptron: esempio

Tr:

es.	$\vec{x}$	target
1	(4,5)	1
2	(6,1)	1
3	(4,1)	-1
4	(1,2)	-1

soglia $\longrightarrow$

es.	$\vec{x}'$	target
1'	(1,4,5)	1
2'	(1,6,1)	1
3'	(1,4,1)	-1
4'	(1,1,2)	-1

vettore dei pesi: $\vec{w} = (w_0, w_1, w_2)$

supponiamo di partire con pesi "random": $\vec{w} = (0, 1, -1)$ e $\eta = \frac{1}{2}$

pesi	input	target	out	errore	nuovi pesi
(0,1,-1)	(1,4,5)	1			
	(1,6,1)	1			
	(1,4,1)	-1			
	(1,1,2)	-1			

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(0,1,-1)	(1,4,5)	1	-1	2	(1,5,4)
(1,5,4)	(1,6,1)	1	1	0	nessun cambiamento
(1,5,4)	(1,4,1)	-1	1	-2	(0,1,3)
(0,1,3)	(1,1,2)	-1	1	-2	(-1,0,1)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-1,0,1)	(1,4,5)	1	1	0	nessun cambiamento
(-1,0,1)	(1,6,1)	1	? (-1)	2	(0,6,2)
(0,6,2)	(1,4,1)	-1	1	-2	(-1,2,1)
(-1,2,1)	(1,1,2)	-1	1	-2	(-2,1,-1)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-2,1,-1)	(1,4,5)	1	-1	2	(-1,5,4)
(-1,5,4)	(1,6,1)	1	1	0	nessun cambiamento
(-1,5,4)	(1,4,1)	-1	1	-2	(-2,1,3)
(-2,1,3)	(1,1,2)	-1	1	-2	(-3,0,1)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-3,0,1)	(1,4,5)	1	1	0	nessun cambiamento
(-3,0,1)	(1,6,1)	1	-1	2	(-2,6,2)
(-2,6,2)	(1,4,1)	-1	1	-2	(-3,2,1)
(-3,2,1)	(1,1,2)	-1	1	-2	(-4,1,-1)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-4,1,-1)	(1,4,5)	1	-1	2	(-3,5,4)
(-3,5,4)	(1,6,1)	1	1	0	nessun cambiamento
(-3,5,4)	(1,4,1)	-1	1	-2	(-4,1,3)
(-4,1,3)	(1,1,2)	-1	1	-2	(-5,0,1)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-5,0,1)	(1,4,5)	1	? (-1)	2	(-4,4,6)
(-4,4,6)	(1,6,1)	1	1	0	nessun cambiamento
(-4,4,6)	(1,4,1)	-1	1	-2	(-5,0,5)
(-5,0,5)	(1,1,2)	-1	1	-2	(-6,-1,3)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-6,-1,3)	(1,4,5)	1	1	0	nessun cambiamento
(-6,-1,3)	(1,6,1)	1	-1	2	(-5,5,4)
(-5,5,4)	(1,4,1)	-1	1	-2	(-6,1,3)
(-6,1,3)	(1,1,2)	-1	1	-2	(-7,0,1)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-7,0,1)	(1,4,5)	1	-1	2	(-6,4,6)
(-6,4,6)	(1,6,1)	1	1	0	nessun cambiamento
(-6,4,6)	(1,4,1)	-1	1	-2	(-7,0,5)
(-7,0,5)	(1,1,2)	-1	1	-2	(-8,-1,3)

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-8,-1,3)	(1,4,5)	1	1	0	nessun cambiamento
(-8,-1,3)	(1,6,1)	1	-1	2	(-7,5,2)
(-7,5,2)	(1,4,1)	-1	1	-2	(-8,1,3)
(-8,1,3)	(1,1,2)	-1	-1	0	nessun cambiamento

Apprendimento per Perceptron: esempio

pesi $\vec{w}$	input $\vec{x}'$	target t	out out	errore $(t-out)$	nuovi pesi $\vec{w} \leftarrow \vec{w} + \frac{1}{2}(t-out)\vec{x}'$
(-8,1,3)	(1,4,5)	1	1	0	nessun cambiamento
(-8,1,3)	(1,6,1)	1	1	0	nessun cambiamento
(-8,1,3)	(1,4,1)	-1	-1	0	nessun cambiamento
(-8,1,3)	(1,1,2)	-1	-1	0	nessun cambiamento

Apprendimento di Reti di Perceptron

Abbiamo visto che un singolo Perceptron non riesce ad apprendere tutte le funzioni booleane (es. XOR)

Però una rete di Perceptron può implementare una qualunque funzione booleana (tramite AND, OR, NOT).

Problema: come effettuare l'apprendimento di una rete di Perceptron ?

Non si sa come assegnare "credito" o "colpe" alle unità nascoste:

PROBLEMA DELL'ASSEGNAMENTO DEL CREDITO

Una possibile soluzione è quella di rendere il singolo neurone derivabile e sfruttare la tecnica di Discesa del Gradiente per apprendere i pesi "giusti".

Vediamo, quindi, come la Discesa del Gradiente si applica ad un Perceptron "semplifi cato".

Discesa di Gradiente

Consideriamo un Perceptron SENZA la hard-threshold:

$$out(\vec{x}) = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$$

e defi niamo una misura dell'errore commesso da un particolare vettore dei pesi:

Funzione Errore: $E[\vec{w}] = \frac{1}{2N_{Tr}} \sum_{(\vec{x}^{(i)}, t^{(i)}) \in Tr} \left(t^{(i)} - out(\vec{x}^{(i)}) \right)^2$

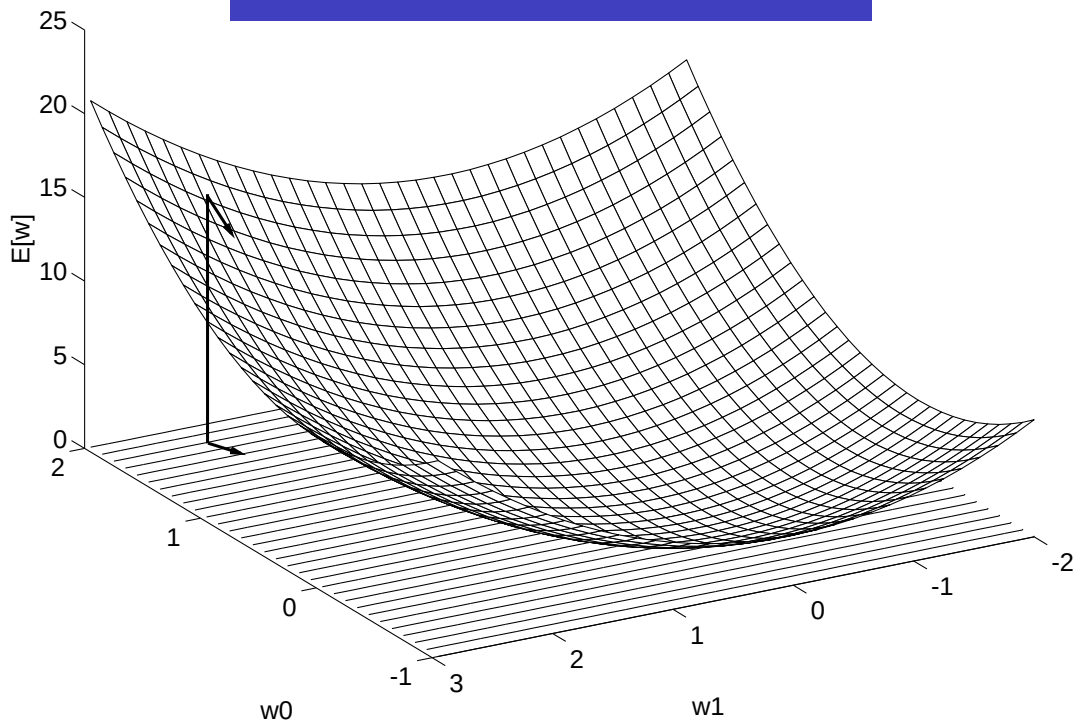
dove N_{Tr} è la cardinalità dell'insieme di apprendimento Tr.

La funzione errore di sopra misura lo scarto quadratico medio (diviso 2) del valore target da quello predetto dal neurone (out).

Ovviamente, se $\forall (\vec{x}^{(i)}, t^{(i)}) \in Tr$ si ha $out(\vec{x}^{(i)}) = t^{(i)} \rightarrow \ E[\vec{w}] = 0$

Quindi bisogna MINIMIZZARE $E[\vec{w}]$ rispetto a $\vec{w}$

Discesa di Gradiente



Idea base: partire da un $\vec{w}$ random e modifi carlo nella direzione contraria al gradiente (che indica la direzione di crescita di $E[\vec{w}]$)

$$\underbrace{\nabla E[\vec{w}]}_{\text{gradiente}} \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right], \quad \Delta\vec{w} = -\eta \nabla E[\vec{w}], \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Calcolo del gradiente

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2N_{Tr}} \sum_{d \in Tr} (t^{(d)} - out^{(d)})^2$$

Calcolo del gradiente

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})^2$$

$$= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)} - out^{(d)})^2$$

Calcolo del gradiente

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)} - out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)} - out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)} - out^{(d)})
\end{aligned}$$

Calcolo del gradiente

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-\vec{w}\cdot\vec{x}^{(d)})
\end{aligned}
$$

Calcolo del gradiente

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-\vec{w}\cdot\vec{x}^{(d)}) \\
\frac{\partial E}{\partial w_i} &= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})(-x_i^{(d)}) \\
&= -\frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})(x_i^{(d)})
\end{aligned}
$$

Discesa di Gradiente

Gradient-Descent(Tr, η)

ogni esempio di apprendimento è una coppia $(\vec{x}, t)$*, dove* $\vec{x}$ *è il vettore di valori in input, e* t *è il valore desiderato in output (target).* η *è il coefficiente di apprendimento (che ingloba* $\frac{1}{N_{Tr}}$*).*

- Assegna a w_i valori piccoli random
- Finché la condizione di terminazione non è verifi cata, fai
  - $\Delta w_i \leftarrow 0$
  - Per ogni $(\vec{x}, t)$ in Tr, fai
    - Presenta $\vec{x}$ al neurone e calcola l'output out
    - Per ogni w_i, fai

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - out)x_i$$

  - Per ogni w_i, fai

$$w_i \leftarrow w_i + \Delta w_i$$

Discesa di Gradiente con Sigmoide

Consideriamo un Perceptron con funzione sigmoidale:

$$out(\vec{x}) = \sigma(\sum_{i=0}^{n} w_i x_i) = \sigma(\vec{w} \cdot \vec{x})$$

dove si ricorda che $\sigma(net) = \frac{1}{1+e^{-net}}$

Si noti che per $\sigma()$ vale la seguente relazione

$$\sigma'(net) = \frac{d\,\sigma(net)}{d\,net} = \sigma(net)(1 - \sigma(net))$$

e ricordiamo che (derivata di funzioni composte)

$$\frac{d\,f(g(x))}{d\,x} = \frac{d\,f(g(x))}{d\,g(x)} \frac{d\,g(x)}{d\,x}$$

Calcolo del gradiente con sigmoide

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})
\end{aligned}$$

Calcolo del gradiente con sigmoide

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-\sigma(\vec{w}\cdot\vec{x}^{(d)}))
\end{aligned}
$$

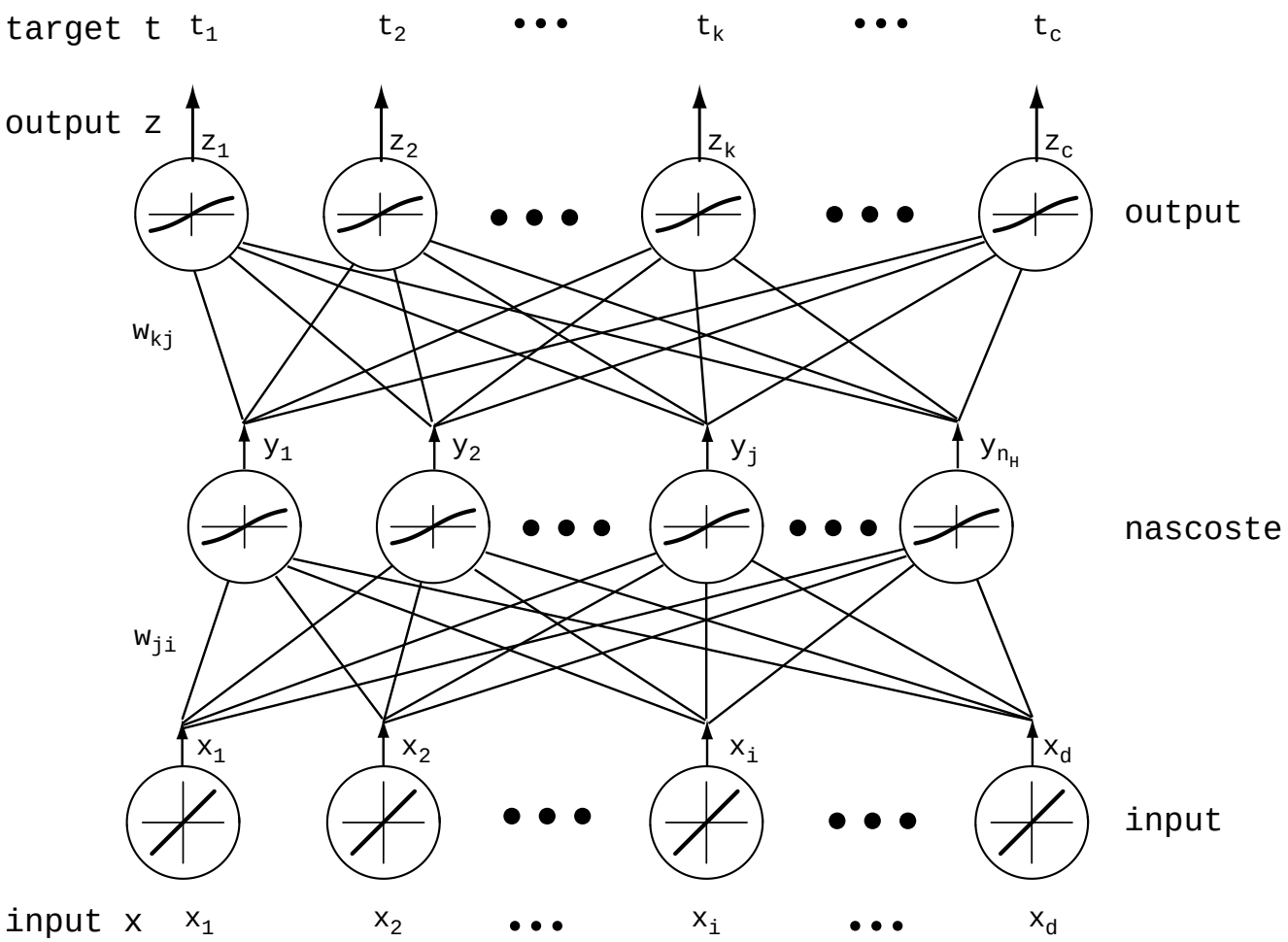
Calcolo del gradiente con sigmoide

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-\sigma(\vec{w}\cdot\vec{x}^{(d)})) \\
\frac{\partial E}{\partial w_i} &= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})(-\frac{\partial\sigma(\vec{w}\cdot\vec{x}^{(d)})}{\partial\vec{w}\cdot\vec{x}^{(d)}}\frac{\partial\vec{w}\cdot\vec{x}^{(d)}}{\partial w_i})
\end{aligned}
$$

Calcolo del gradiente con sigmoide

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)})^2 \\
&= \frac{1}{2N_{Tr}}\sum_{d\in Tr}2(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-out^{(d)}) \\
&= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\frac{\partial}{\partial w_i}(t^{(d)}-\sigma(\vec{w}\cdot\vec{x}^{(d)})) \\
\frac{\partial E}{\partial w_i} &= \frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})(-\frac{\partial\sigma(\vec{w}\cdot\vec{x}^{(d)})}{\partial\vec{w}\cdot\vec{x}^{(d)}}\frac{\partial\vec{w}\cdot\vec{x}^{(d)}}{\partial w_i}) \\
&= -\frac{1}{N_{Tr}}\sum_{d\in Tr}(t^{(d)}-out^{(d)})\sigma(\vec{w}\cdot\vec{x}^{(d)})(1-\sigma(\vec{w}\cdot\vec{x}^{(d)}))x_i^{(d)}
\end{aligned}
$$

Reti Neurali Feed-forward: notazione



Reti Neurali Feed-forward: notazione

- d unità di ingresso, dimensione dei dati in ingresso $\vec{x} \equiv (x_1, \ldots, x_d)$
  ($d+1$ se si include la soglia nel vettore dei pesi $\vec{x}' \equiv (x_0, x_1, \ldots, x_d)$)
- N_H unità nascoste (con output $\vec{y} \equiv (y_1, \ldots, y_{N_H})$)
- c unità di output, dimensione dei dati in output $\vec{z} \equiv (z_1, \ldots, z_c)$
- c, dimensione dei dati desiderati $\vec{t} \equiv (t_1, \ldots, t_c)$
- w_{ji} peso dalla unità di ingresso i alla unità nascosta j
- w_{kj} peso dalla unità nascosta j alla unità di output k

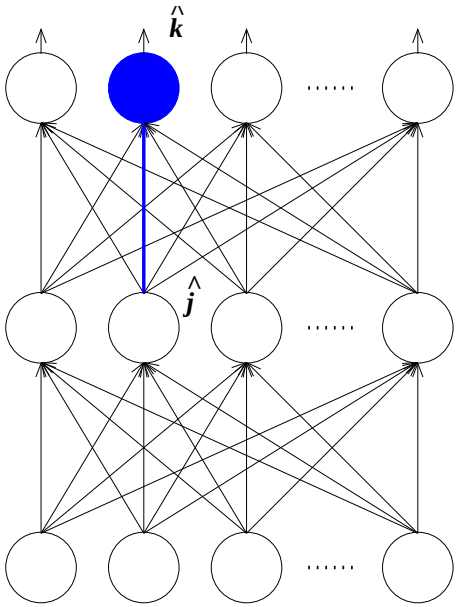
La funzione errore, considerando che si hanno c unità di output, diventa

$$E[\vec{w}] = \frac{1}{2cN_{Tr}} \sum_{(\vec{x}^{(p)}, \vec{t}^{(p)}) \in Tr} \sum_{k=1}^{c} \left(t_k^{(p)} - z_k(\vec{x}^{(p)}) \right)^2$$

Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

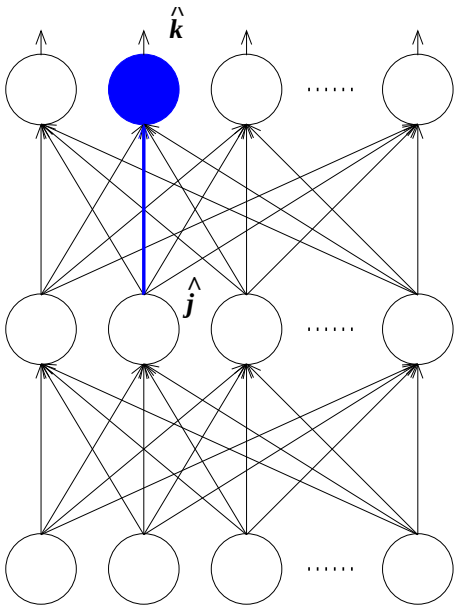


Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$



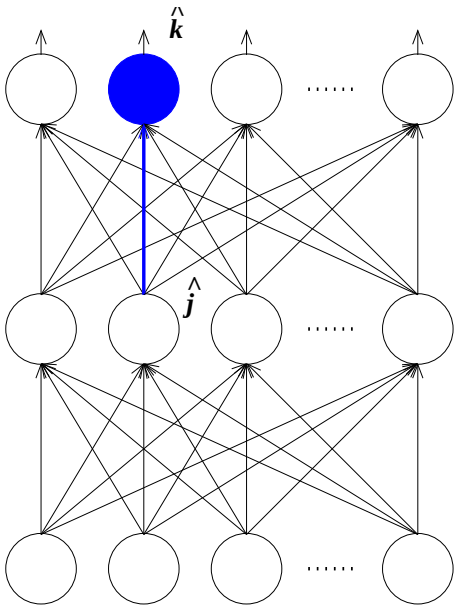
Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$



Calcolo del gradiente per i pesi di una unità di output

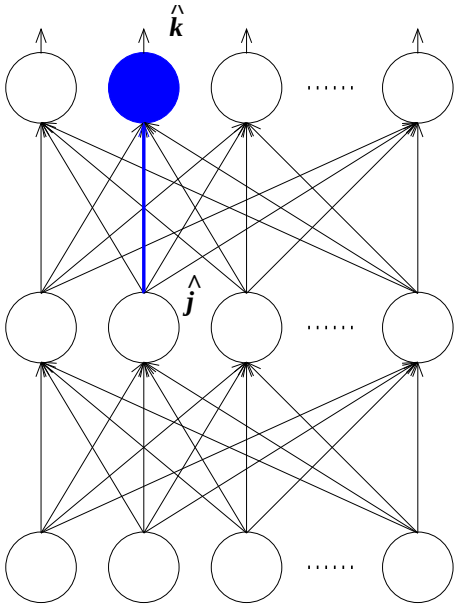
Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$

$$= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}))$$



Calcolo del gradiente per i pesi di una unità di output

Fissiamo gli indici $\hat{k}$ e $\hat{j}$:

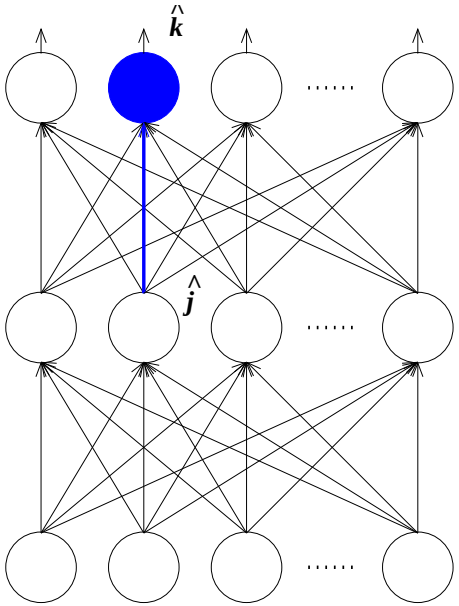
$$\frac{\partial E}{\partial w_{\hat{k}\hat{j}}} = \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \frac{\partial}{\partial w_{\hat{k}\hat{j}}} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} 2(t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)})$$

$$= \frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \frac{\partial}{\partial w_{\hat{k}\hat{j}}} (t_{\hat{k}}^{(p)} - \sigma(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}))$$

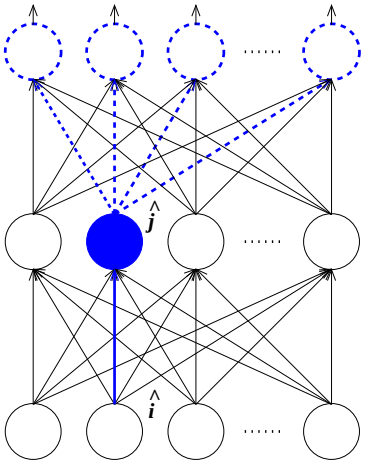
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} (t_{\hat{k}}^{(p)} - z_{\hat{k}}^{(p)}) \sigma'(\vec{w}_{\hat{k}} \cdot \vec{y}^{(p)}) y_{\hat{j}}^{(p)}$$



Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

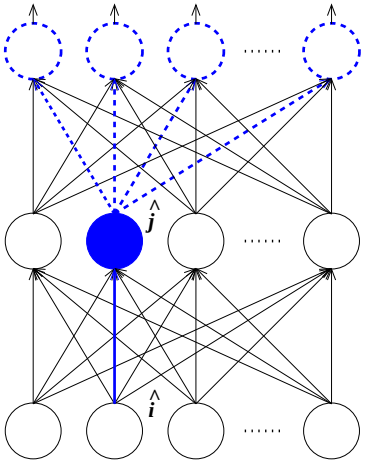


Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



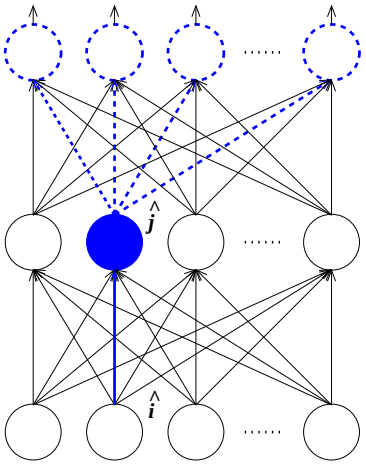
Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$



Calcolo del gradiente per i pesi di una unità nascosta

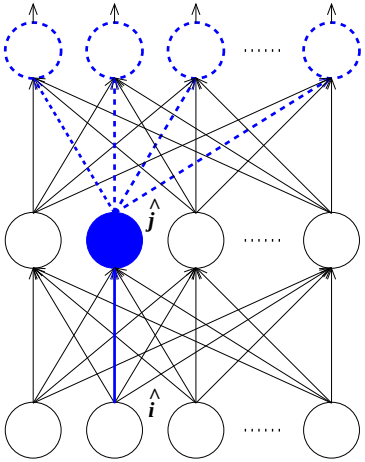
Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \vec{w}_k \cdot \vec{y}^{(p)}$$

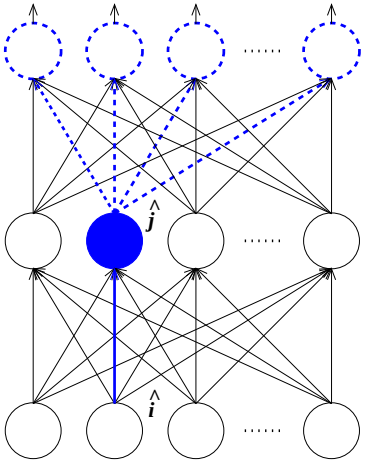


Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

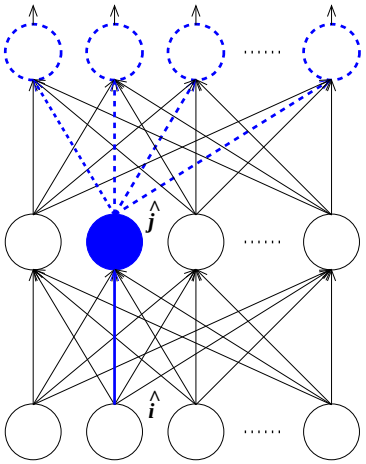
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (t_k^{(p)} - z_k^{(p)})^2$$



$$= \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

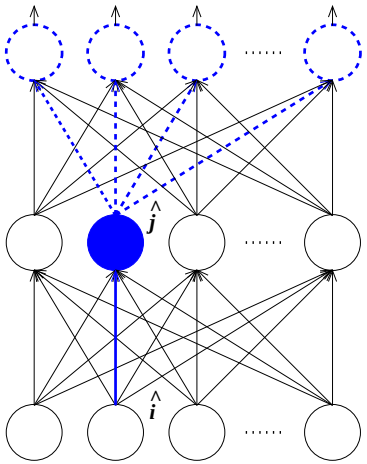
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$



$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

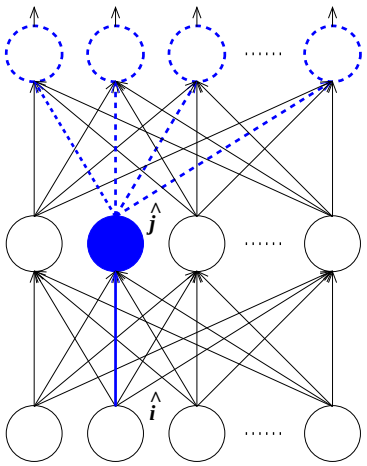
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)})$$

Calcolo del gradiente per i pesi di una unità nascosta

Fissiamo gli indici $\hat{j}$ e $\hat{i}$:

$$\frac{\partial E}{\partial w_{\hat{j}\hat{i}}} = \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} 2(t_k^{(p)} - z_k^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (-z_k^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} \sum_{j=1}^{N_H} w_{kj} y_j^{(p)}$$

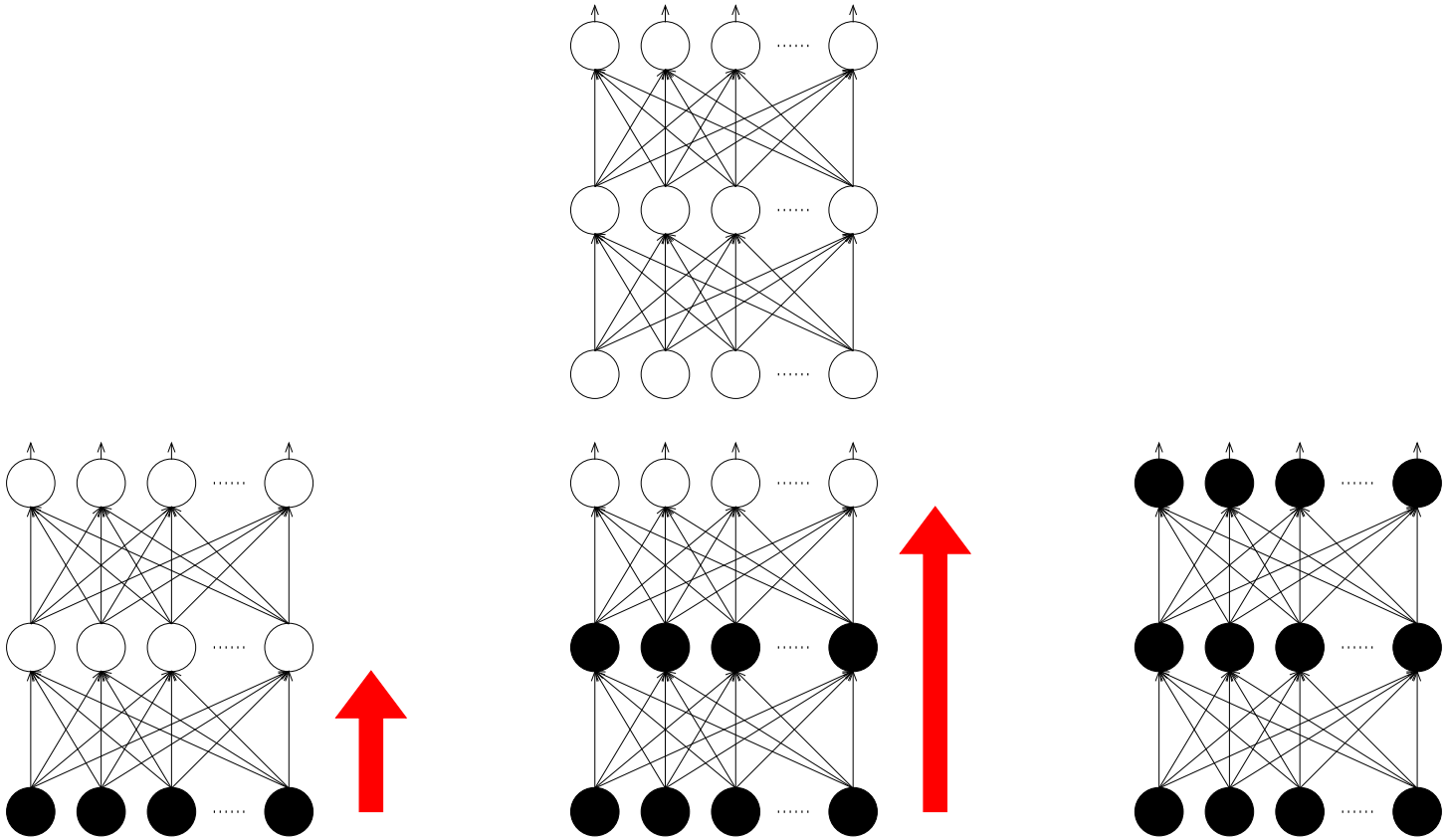


$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \frac{\partial}{\partial w_{\hat{j}\hat{i}}} y_{\hat{j}}^{(p)}$$

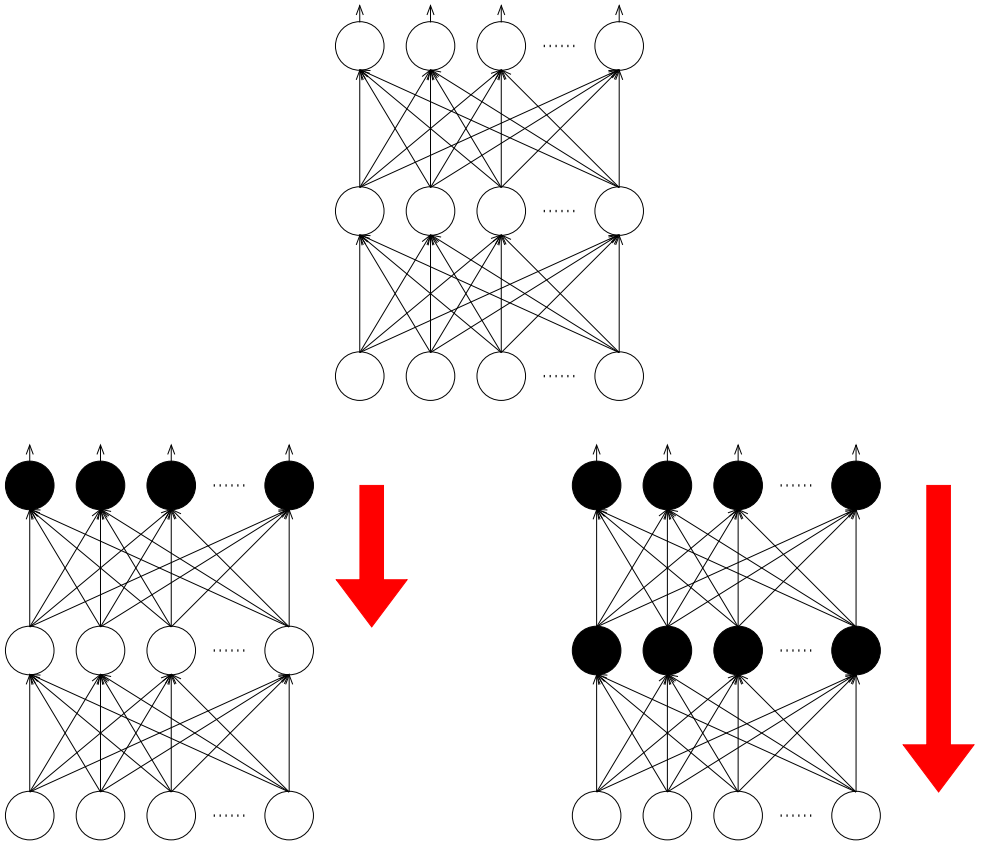
$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) \frac{\partial}{\partial w_{\hat{j}\hat{i}}} (\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)})$$

$$= -\frac{1}{cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)}) \sigma'(\vec{w}_k \cdot \vec{y}^{(p)}) w_{k\hat{j}} \sigma'(\vec{w}_{\hat{j}} \cdot \vec{x}^{(p)}) x_{\hat{i}}^{(p)}$$

Reti Neurali Feed-forward: Fase Forward



Reti Neurali Feed-forward: Fase Backward



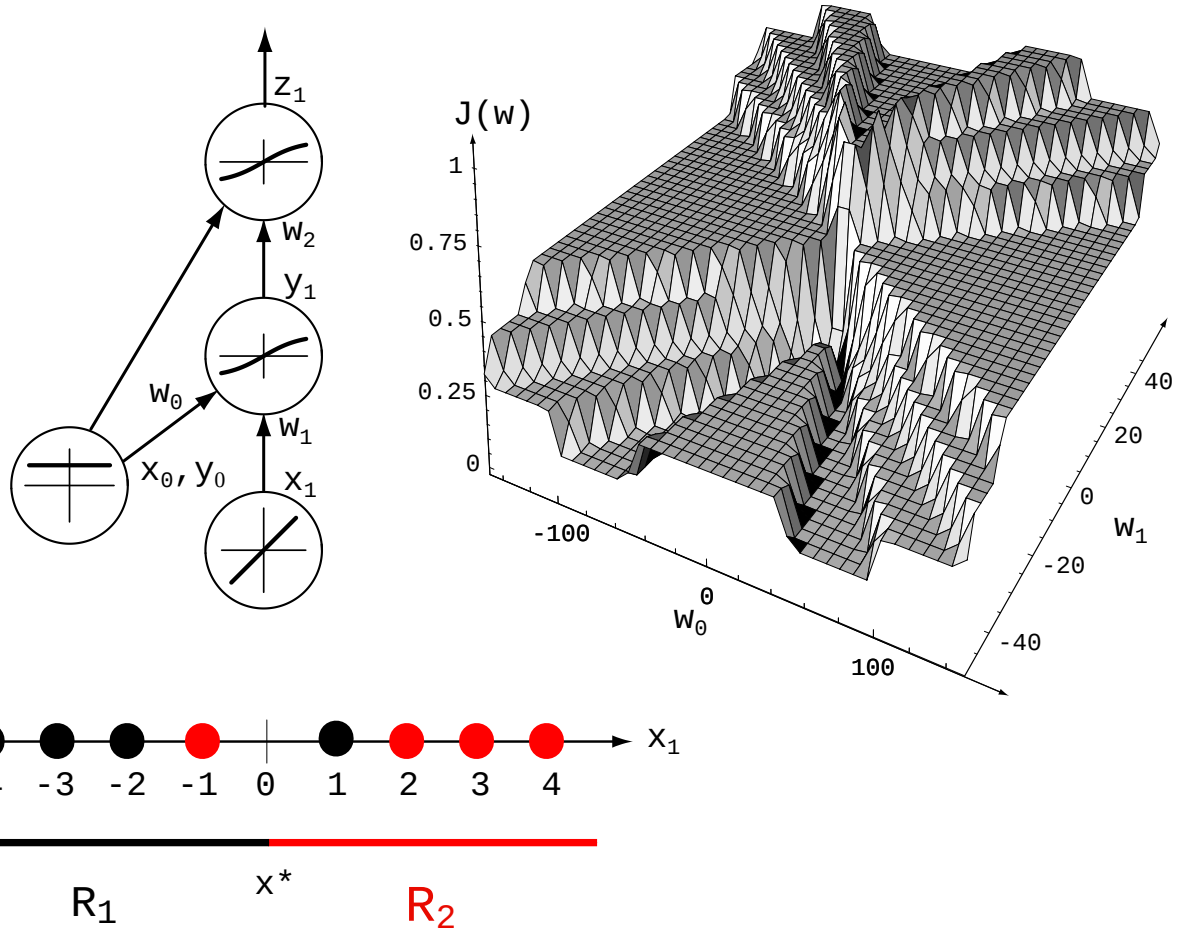
Algoritmo Back-Propagation (uno strato nascosto, stocastico)

Back-Propagation-1hl-stocastico(Tr, η, *topologia rete*)

- Inizializza tutti i pesi a valori random piccoli
- Finché la condizione di terminazione non è verifi cata, fai
  - Per ogni $(\vec{x}, \vec{t})$ in Tr, fai
    1. presenta $\vec{x}$ alla rete e calcola il corrispondente output
    2. Per ogni unità di output k
    $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
    3. Per ogni unità nascosta j
    $$\delta_j \leftarrow o_j(1 - o_j) \sum_{k \in outputs} w_{k,j}\delta_k$$
    4. aggiorna tutti i pesi $w_{p,q}$ della rete

$$w_{s,q} \leftarrow w_{s,q} + \eta \Delta w_{s,q} \quad \text{dove} \quad \Delta w_{s,q} = \begin{cases} \delta_s x_q & \text{se } s \in nascoste \\ \delta_s y_q & \text{se } s \in outputs \end{cases}$$

Esempio di Funzione Errore



Discesa di Gradiente Batch e Stocastica

Batch:

Fai finché condizione di terminazione non soddisfatta

1. calcola $\nabla E_{Tr}[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{Tr}[\vec{w}]$

Stocastica (Incrementale):

Fai finché condizione di terminazione non soddisfatta

- Per ogni esempio di apprendimento p in Tr
  1. calcola $\nabla E_{p \in Tr}[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_{p \in Tr}[\vec{w}]$

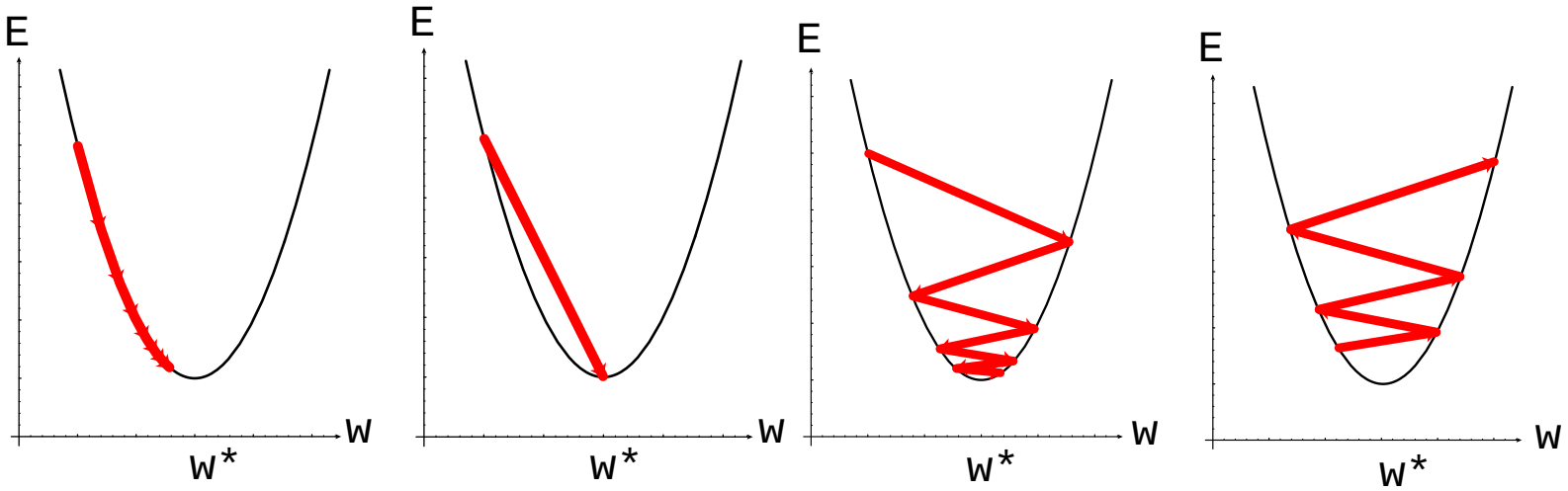
dove

$$E_{Tr}[\vec{w}] \equiv \frac{1}{2cN_{Tr}} \sum_{p \in Tr} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2 \qquad E_{p \in Tr}[\vec{w}] \equiv \frac{1}{2c} \sum_{k=1}^{c} (t_k^{(p)} - z_k^{(p)})^2$$

La discesa di gradiente *Stocastica* (gradiente istantaneo) può approssimare quella *Batch* (gradiente esatto) con precisione arbitraria se η è suffi cientemente piccolo

Alcuni Problemi ...

- Scelta della topologia della rete $\rightarrow$ determina lo Spazio delle Ipotesi;
- Scelta del passo di discesa (valore di η):

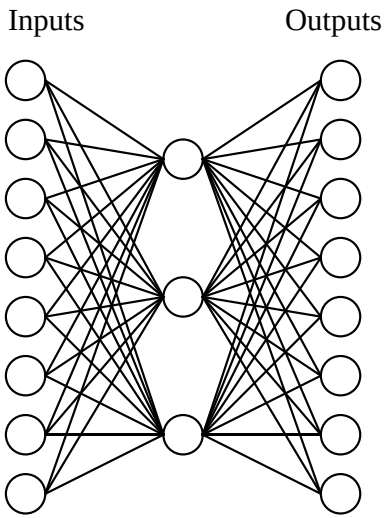


- apprendimento lento..., ma calcolo di output veloce
- MINIMI LOCALI !!

Bias Induttivo: sia nella rappresentazione che nella ricerca

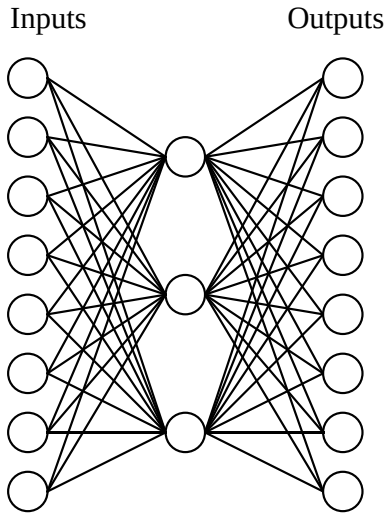
Esempio di Apprendimento per Rete Feed-forward

Compressione di Dati



Input	Output
00000001	00000001
00000010	00000010
00000100	00000100
00001000	00001000
00010000	00010000
00100000	00100000
01000000	01000000
10000000	10000000

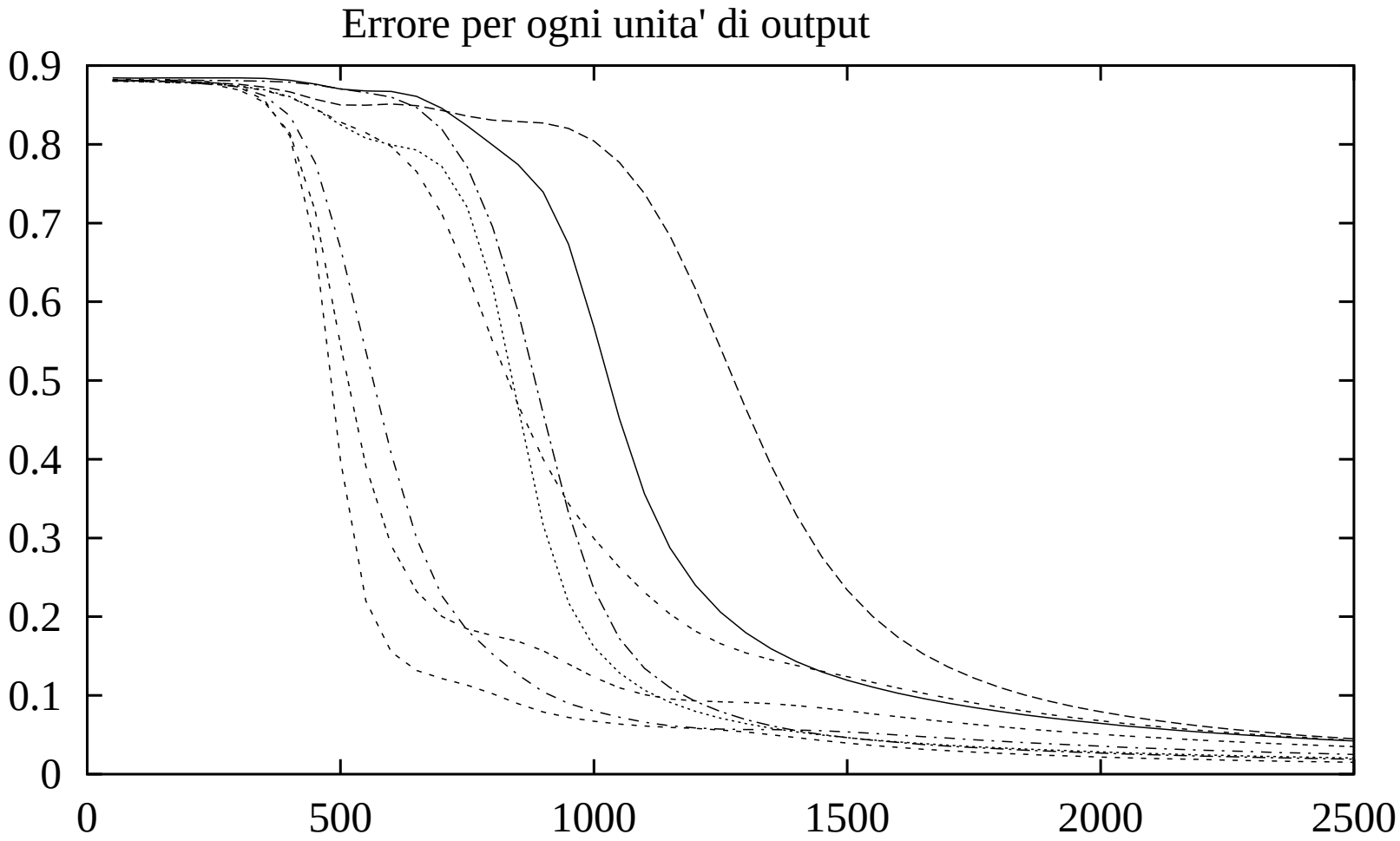
Esempio di Apprendimento per Rete Feed-forward



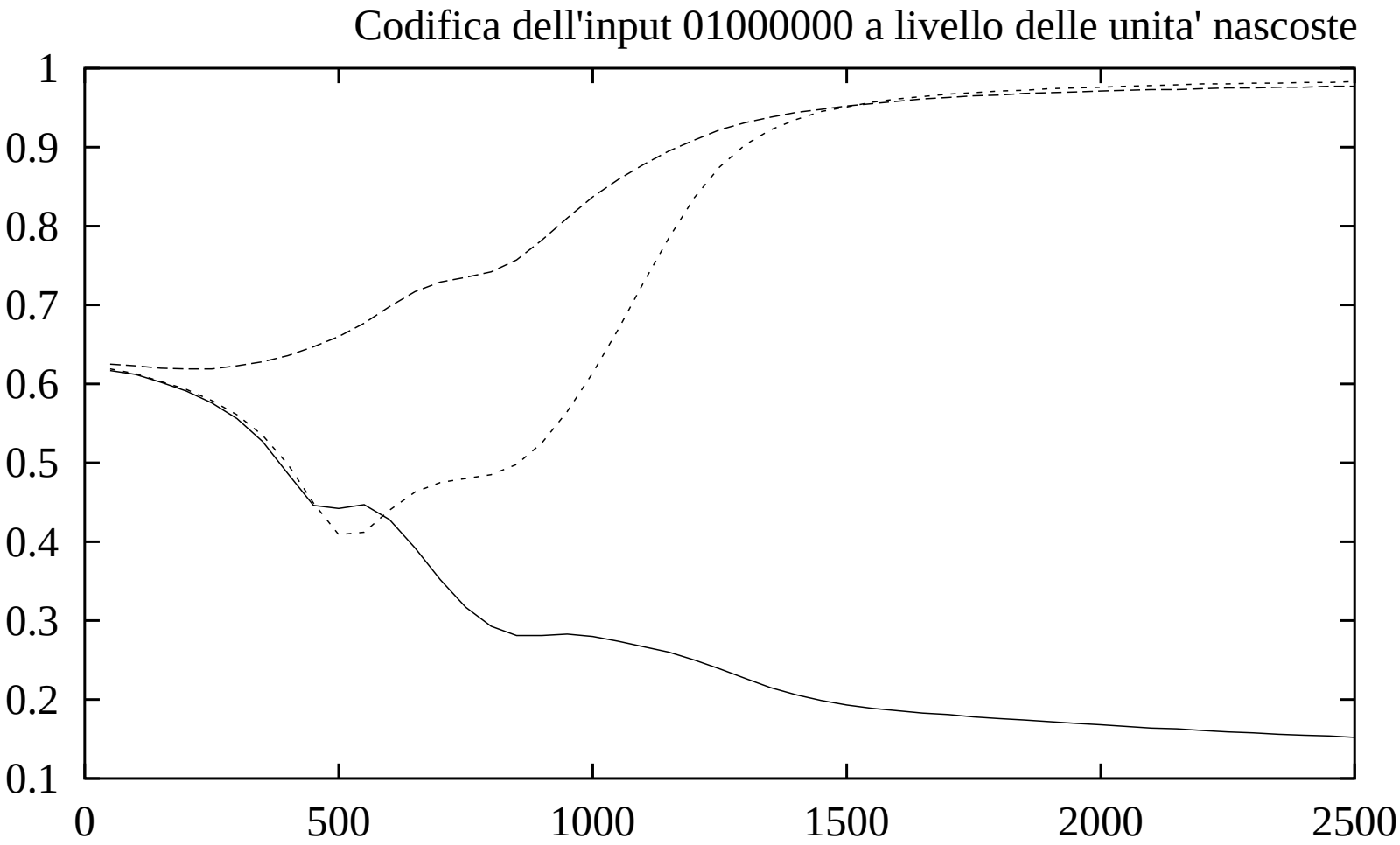
Compressione di Dati

Input		Valori Nascosti		Output
10000000	$\rightarrow$	0.89 0.04 0.08	$\rightarrow$	10000000
01000000	$\rightarrow$	0.01 0.11 0.88	$\rightarrow$	01000000
00100000	$\rightarrow$	0.01 0.97 0.27	$\rightarrow$	00100000
00010000	$\rightarrow$	0.99 0.97 0.71	$\rightarrow$	00010000
00001000	$\rightarrow$	0.03 0.05 0.02	$\rightarrow$	00001000
00000100	$\rightarrow$	0.22 0.99 0.99	$\rightarrow$	00000100
00000010	$\rightarrow$	0.80 0.01 0.98	$\rightarrow$	00000010
00000001	$\rightarrow$	0.60 0.94 0.01	$\rightarrow$	00000001

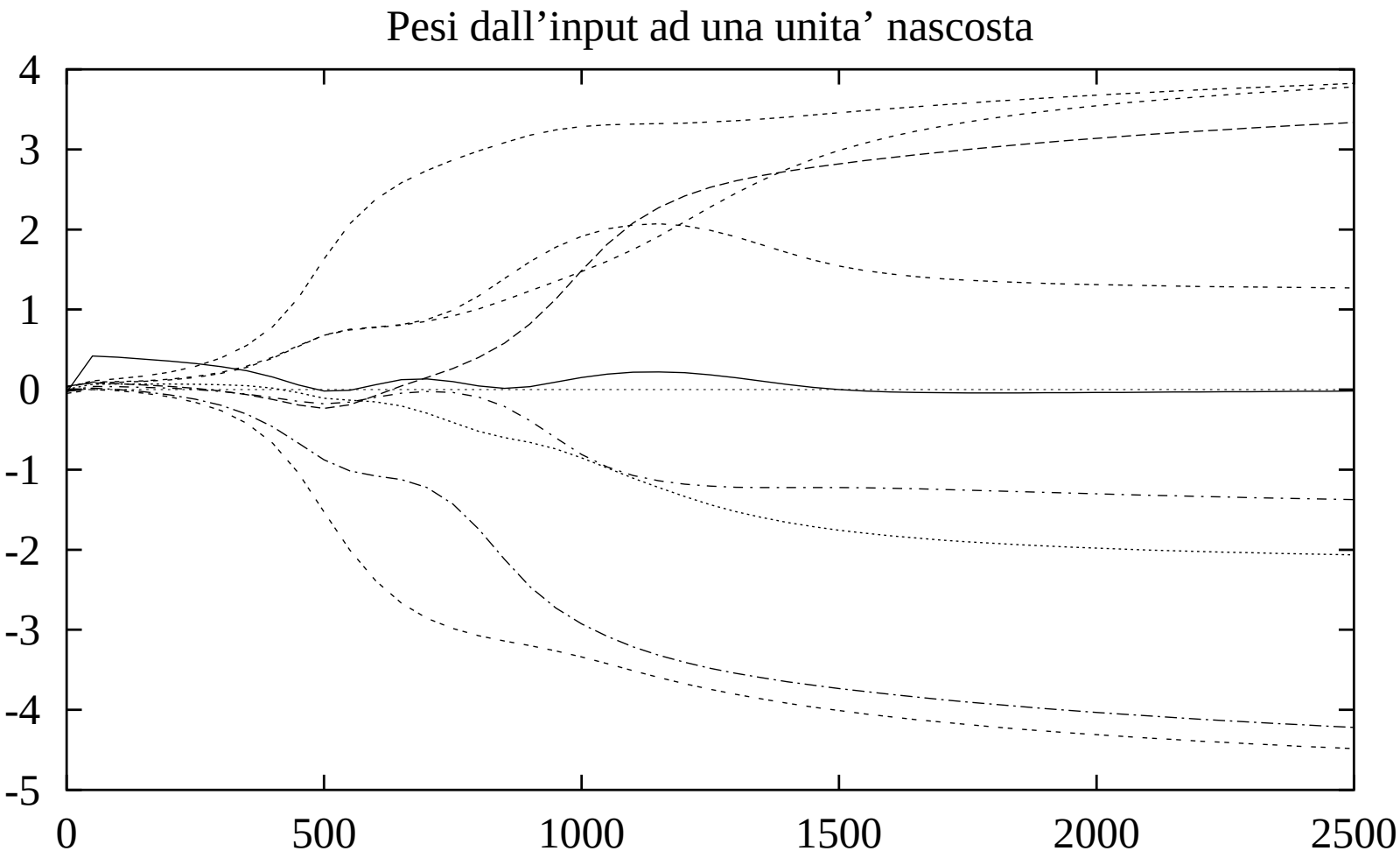
Curve di Apprendimento



Curve di Apprendimento



Curve di Apprendimento



Potere Computazionale Reti Neurali

Il seguente teorema stabilisce l'universalità di reti feed-forward come approssimatori di funzioni continue.

Teorema Sia $\varphi(\cdot)$ una funzione continua monotona crescente, limitata e noncostante. Si indichi con I_n l'ipercubo n-dimensionale $[0,1]^n$ e lo spazio delle funzioni continue su esso defi nite sia $C(I_n)$. Data una qualunque funzione $f \in C(I_n)$ e $\varepsilon > 0$, allora esiste un intero M e insiemi di costanti reali α_i, θ_i, e w_{ij}, dove $i = 1, \ldots, M$ e $j = 1, \ldots, n$ tale che $f(\cdot)$ possa essere approssimata da

$$F(x_1, \ldots, x_n) = \sum_{i=1}^{M} \alpha_i \varphi(\sum_{j=1}^{n} w_{ij} x_j - \theta_i) \tag{1}$$

in modo tale che

$$|F(x_1, \ldots, x_n) - f(x_1, \ldots, x_n)| < \varepsilon \tag{2}$$

per tutti i punti $[x_1, \ldots, x_n] \in I_n$.

Potere Computazionale Reti Neurali

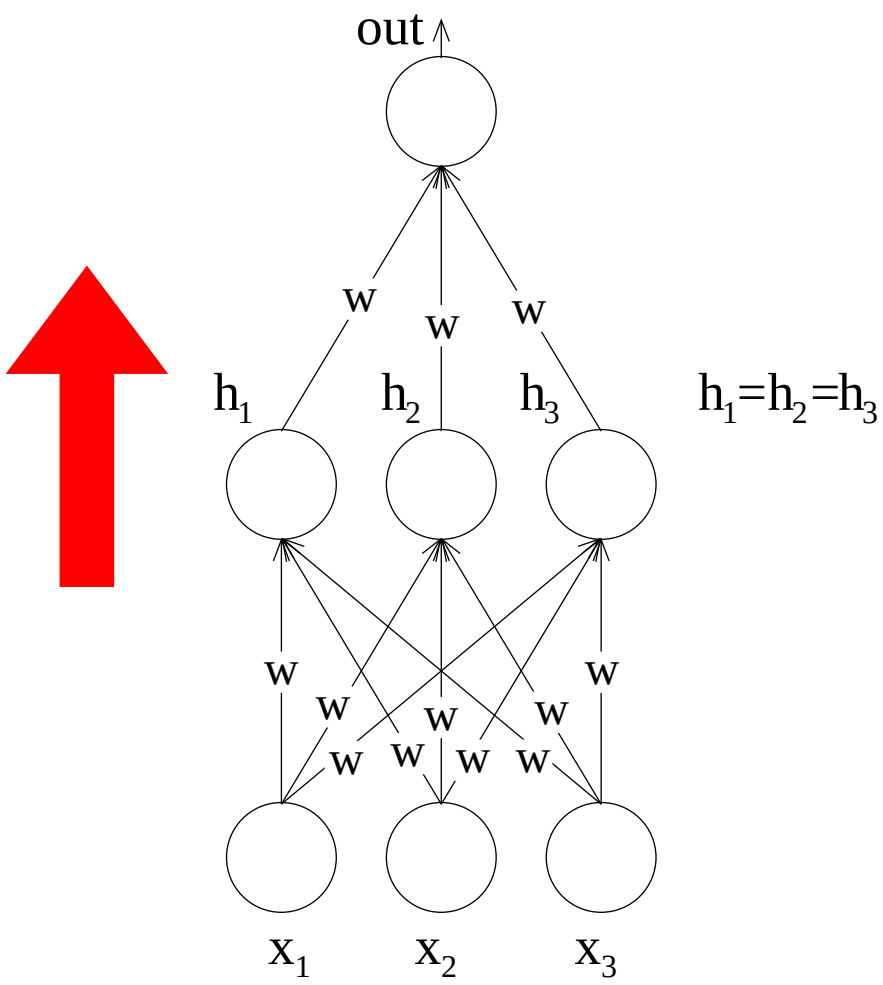
Notare che qualunque funzione sigmoidale soddisfa le condizioni imposte nel teorema su $\varphi(\cdot)$. Inoltre, l'equazione (1) rappresenta l'output di una rete multistrato descritta come segue

1. la rete ha n nodi di input ed un singolo strato di unità nascoste con M unità; gli input sono denotati da $x_1, \ldots, x_n$.
2. l'i-esima unità ha associati i pesi $w_{i1}, \ldots, w_{in}$ e soglia θ_i.
3. l'output della rete è una combinazione lineare degli output delle unità nascoste, dove i coeffi cienti della combinazione sono dati da $\alpha_1, \ldots, \alpha_M$.

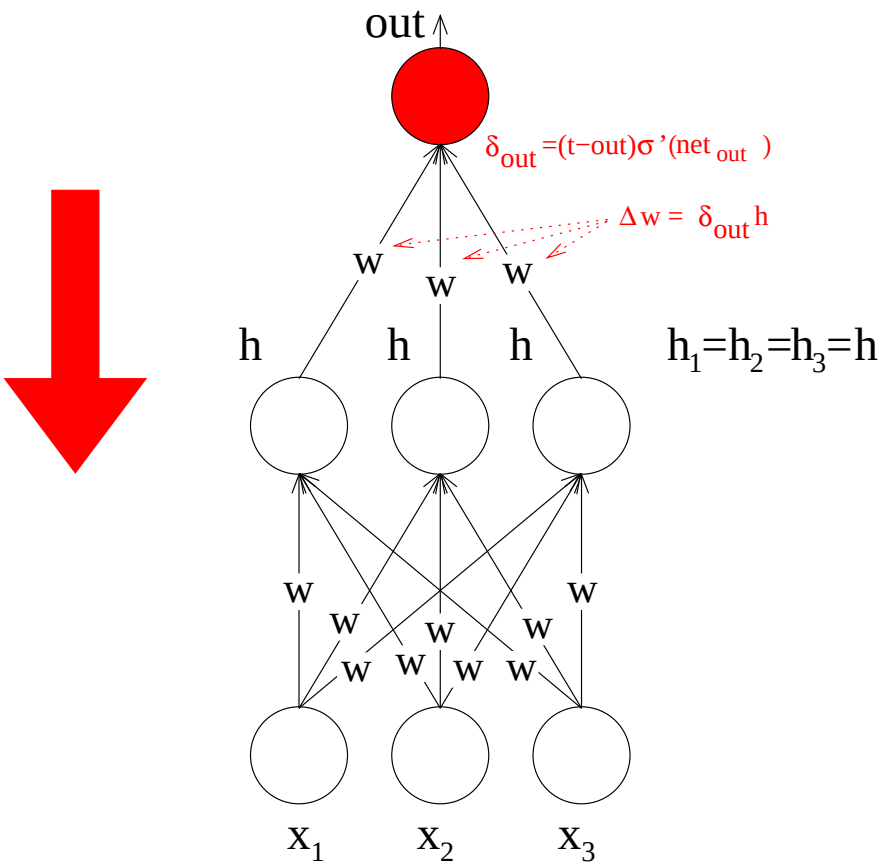
Quindi, data una tolleranza ε, una rete con un unico strato nascosto può approssimare una qualsiasi funzione in $C(I_n)$.

Si noti che il teorema afferma solo l'esistenza di una rete e non fornisce alcuna formula per il calcolo del numero M di unità nascoste necessarie per approssimare la funzione target con la tolleranza desiderata.

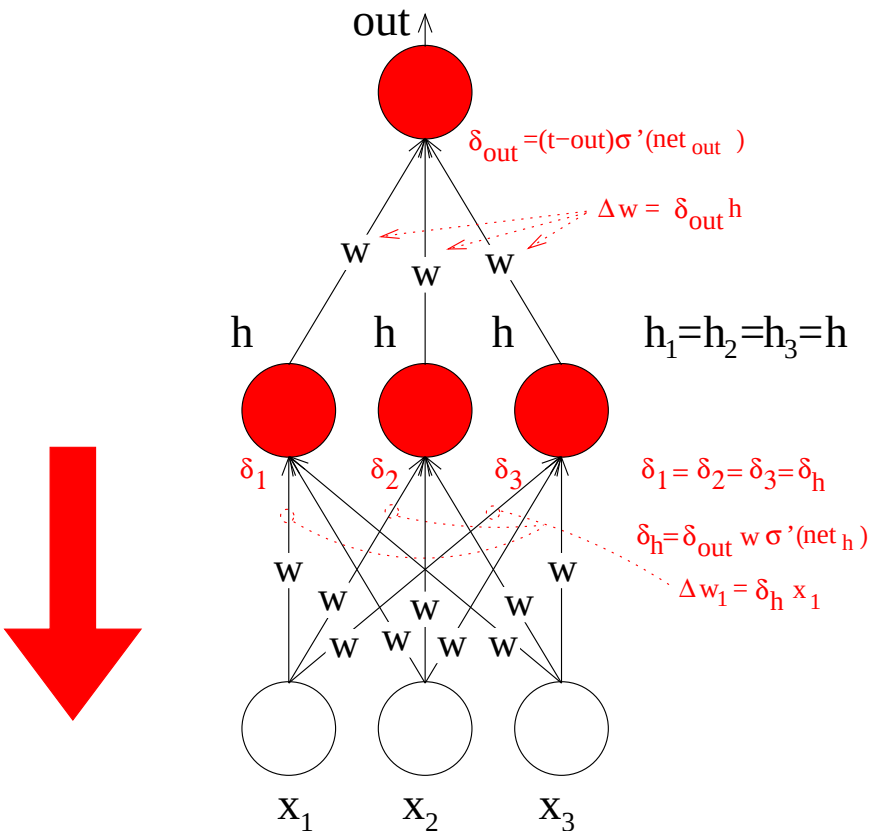
Simmetrie



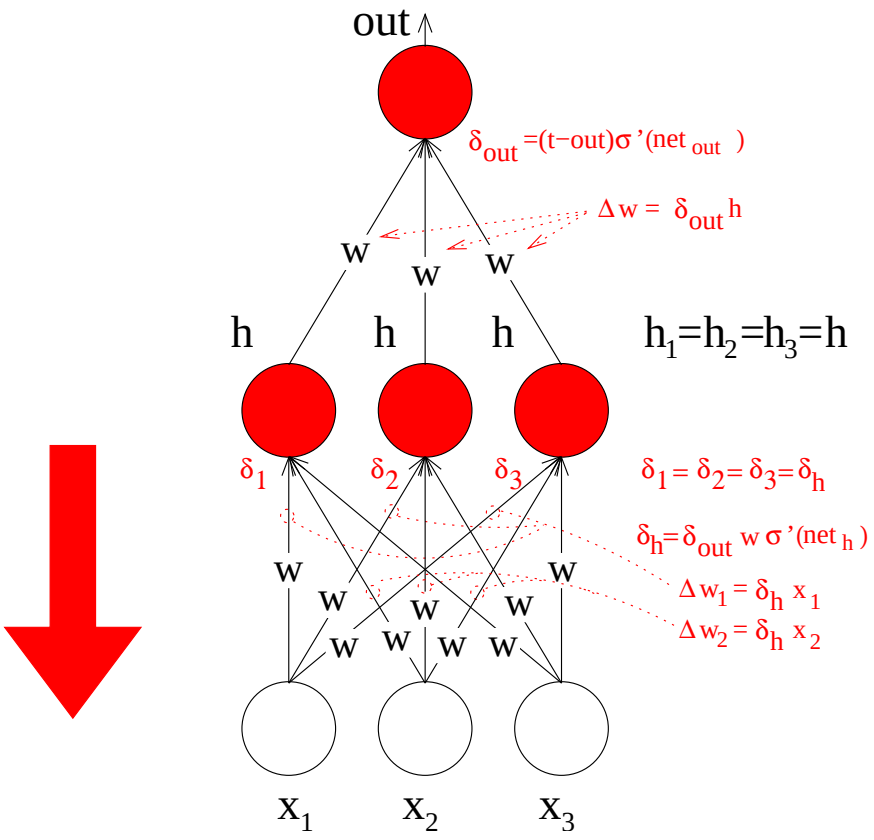
Simmetrie


out
$\delta_{out} = (t - out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
x_1 x_2 x_3

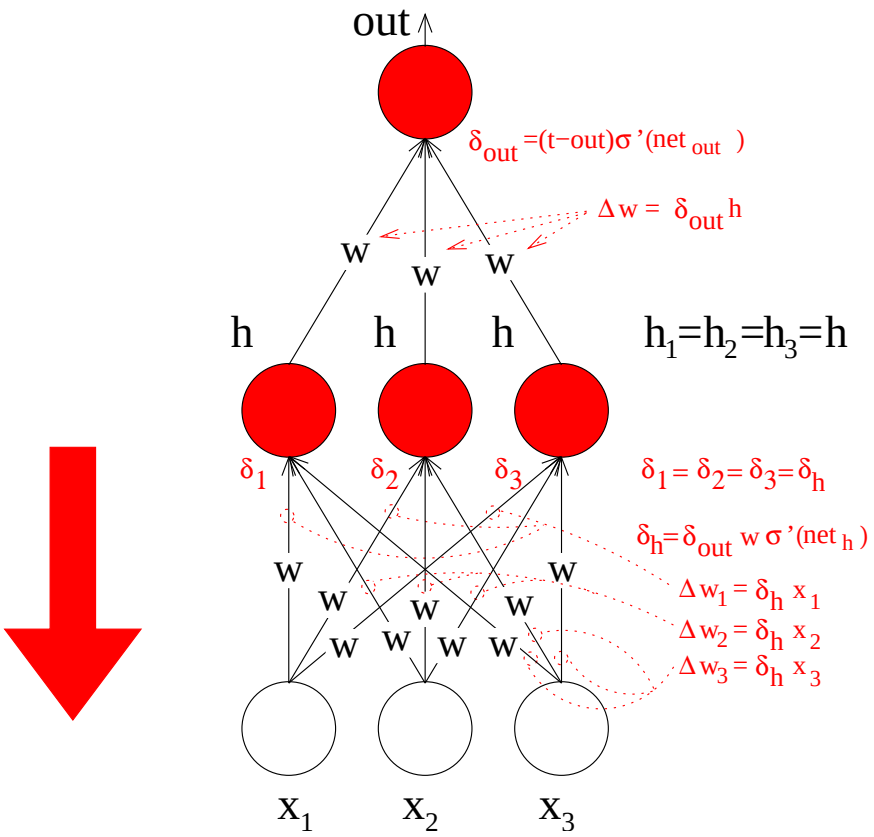
Simmetrie


out
$\delta_{out} = (t-out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
δ_1 δ_2 δ_3
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out} w \sigma'(net_h)$
$\Delta w_1 = \delta_h x_1$
x_1 x_2 x_3

Simmetrie


out
$\delta_{out}=(t-out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1=h_2=h_3=h$
δ_1 δ_2 δ_3
$\delta_1=\delta_2=\delta_3=\delta_h$
$\delta_h=\delta_{out}\, w\, \sigma'(net_h)$
$\Delta w_1 = \delta_h x_1$
$\Delta w_2 = \delta_h x_2$
x_1 x_2 x_3

Simmetrie


out
$\delta_{out} = (t-out)\sigma'(net_{out})$
$\Delta w = \delta_{out} h$
w
h
$h_1 = h_2 = h_3 = h$
δ_1 δ_2 δ_3
$\delta_1 = \delta_2 = \delta_3 = \delta_h$
$\delta_h = \delta_{out} w \sigma'(net_h)$
$\Delta w_1 = \delta_h x_1$
$\Delta w_2 = \delta_h x_2$
$\Delta w_3 = \delta_h x_3$
x_1 x_2 x_3

Apprendimento Perceptron: prova di convergenza

Sia dato un insieme di esempi $Tr = \{(\vec{x}^{(i)}, t^{(i)})\}$, dove $t \in \{-1, +1\}$ (classificazione binaria)

Se $\exists \vec{w}^*$ t.c. (lineare separabilità)

$$\forall i,\ t^{(i)}(\vec{w}^* \cdot \vec{x}^{(i)}) \geq \delta = \min_i\ t^{(i)}(\vec{w}^* \cdot \vec{x}^{(i)}) > 0$$

allora

$$\forall i,\ \vec{w}^* \cdot (t^{(i)}\vec{x}^{(i)}) \geq \delta$$

cioè $\vec{w}^*$ è soluzione anche del problema di apprendimento definito dall'insieme di apprendimento $Tr' = \{(t^{(i)}\vec{x}^{(i)}, +1)\}$ e viceversa

Quindi ci possiamo concentrare su problemi dove **tutti gli esempi sono positivi**

Apprendimento Perceptron: prova di convergenza

Supponiamo che

- $\vec{w} = \vec{w}(0) = 0$ (inizializzazione)
- $\eta = \frac{1}{2}$ e $\forall i,\ \|\vec{x}^{(i)}\|^2 \leq K$

Dopo aver commesso q errori (tutti falsi negativi) si avrà

$$\vec{w}(q) = \sum_{j=1}^{q} \vec{x}^{(i_j)}$$

infatti all' errore j-esimo il vettore dei pesi è aggiornato sommandogli l'input $\vec{x}^{(i_j)}$ classificato erroneamente:

$$\vec{w}(j) = \vec{w}(j-1) + \vec{x}^{(i_j)}$$

Apprendimento Perceptron: prova di convergenza

Mostriamo adesso che il modulo di $\vec{w}(q)$ non può crescere indefinitivamente (succede se non si converge ad una soluzione in un numero finito di iterazioni)

Iniziamo definendo un lower bound sul modulo di $\vec{w}(q)$:

$$\vec{w}^* \cdot \vec{w}(q) = \vec{w}^* \sum_{j=1}^{q} \vec{x}^{(i_j)} \geq q\delta \quad \text{(ricordiamo che } \delta = \min_i \ \vec{w}^* \cdot \vec{x}^{(i)}\text{)}$$

e per la disuguaglianza di Cauchy-Swartz ($[\vec{x} \cdot \vec{y}]^2 \leq \|\vec{x}\|^2 \|\vec{y}\|^2$) abbiamo

$$\|\vec{w}^*\|^2 \|\vec{w}(q)\|^2 \geq [\vec{w}^* \cdot \vec{w}(q)]^2 \geq [q\delta]^2 \ \Rightarrow \ \|\vec{w}(q)\|^2 \geq \frac{[q\delta]^2}{\|\vec{w}^*\|^2}$$

Apprendimento Perceptron: prova di convergenza

Definiamo adesso un upper bound sul modulo di $\vec{w}(q)$:

$$\|\vec{w}(q)\|^2 = \|\vec{w}(q-1)+\vec{x}^{(i_q)}\|^2 = \|\vec{w}(q-1)\|^2+2\vec{w}(q-1)\cdot\vec{x}^{(i_q)}+\|\vec{x}^{(i_q)}\|^2$$

e poiché $\vec{w}(q-1)\cdot\vec{x}^{(i_q)} < 0$ (q-esimo errore)

$$\|\vec{w}(q)\|^2 \leq \|\vec{w}(q-1)\|^2 + \|\vec{x}^{(i_q)}\|^2$$

Applicando ricorsivamente questa disuguaglianza su tutti gli errori abbiamo

$$\|\vec{w}(q)\|^2 \leq \sum_{i=1}^{q} \|\vec{x}^{(i_q)}\|^2 \leq qK$$

Apprendimento Perceptron: prova di convergenza

Mettendo insieme il lower bound con l'upper bound otteniamo:

$$\frac{[q\delta]^2}{||\vec{w}^*||^2} \leq ||\vec{w}(q)||^2 \leq qK$$

Pertanto il numero massimo di errori q_{max} che si possono commettere mantenendo i due vincoli soddisfatti è ottenuto quando vale l'uguaglianza dei bound:

$$\frac{[q_{max}\delta]^2}{||\vec{w}^*||^2} = q_{max}K$$

da cui si ottiene

$$q_{max} = \frac{||\vec{w}^*||^2 K}{\delta^2}$$

Quindi, in caso di lineare separabilità, l'algoritmo di apprendimento del Perceptron commette un numero finito di errori, al più $\frac{||\vec{w}^*||^2 K}{\delta^2}$

Structural Risk Minimization

Problema: all'aumentare della VC-dimension diminuisce l'errore empirico (termine A), ma aumenta la VC confi dence (termine B)!

L'approccio Structural Risk Minimization tenta di trovare un compromesso tra i due termini:

Si considerano $\mathcal{H}_i$ tali che

- $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \cdots \subseteq \mathcal{H}_n$
- $VC(\mathcal{H}_1) \leq \cdots \leq VC(\mathcal{H}_n)$
- si seleziona l'ipotesi che ha il bound sull'errore ideale pi`u basso

Esempio: Reti neurali con un numero crescente di neuroni nascosti